CONTROL DATA INSTITUTE

CONTROL DATA CORPORATION

6600 CENTRAL PROCESSOR

Volume I

CONTROL & MEMORY

6600 CENTRAL PROCESSOR

Volume I

Control and Memory

FOR TRAINING PURPOSES ONLY

This book was compiled and written by members of the instructional staff of

CONTROL DATA INSTITUTE
CONTROL DATA CORPORATION

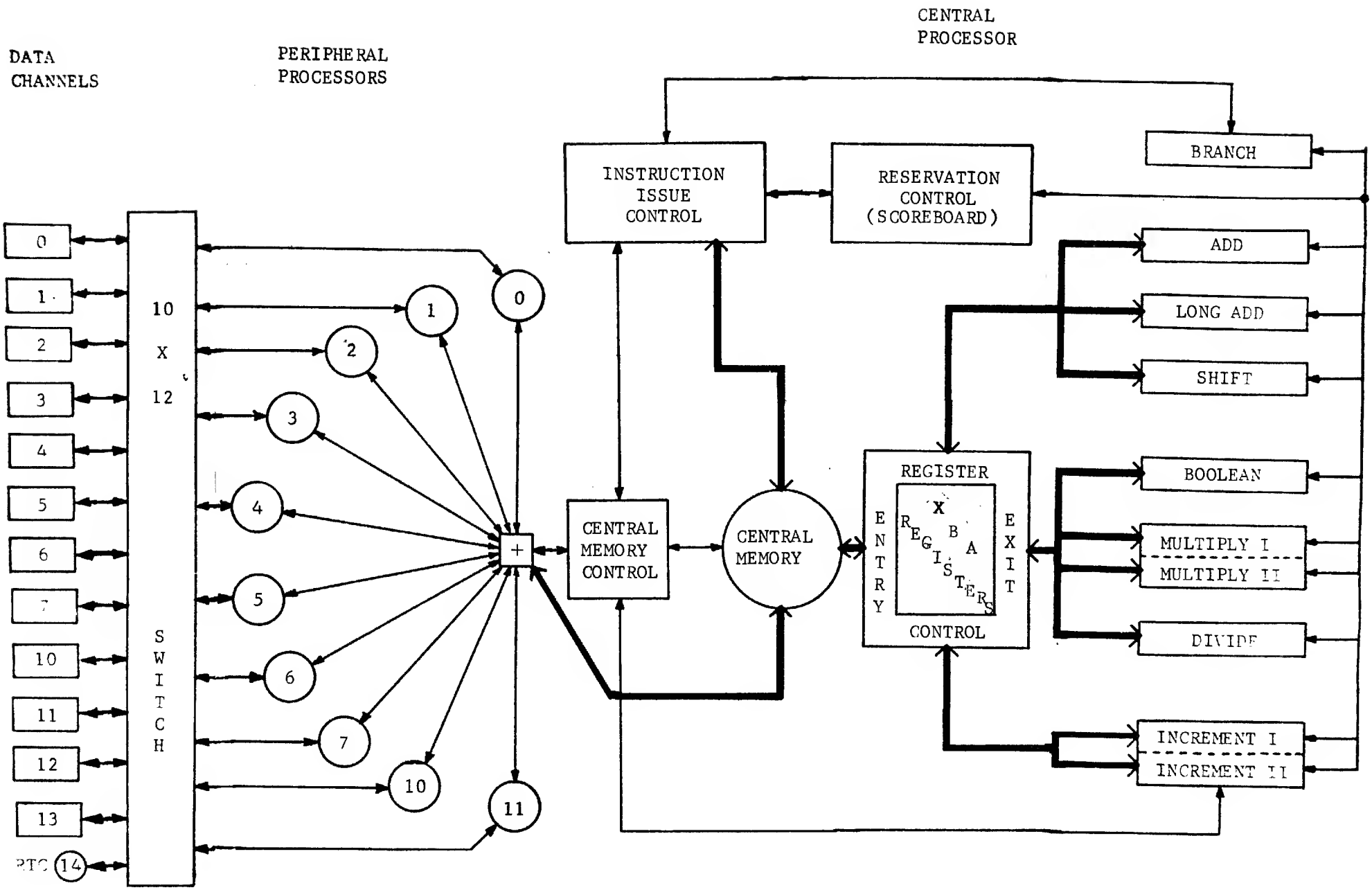
Publication No. 020167

March, 1967

CONTENTS

CHAPTER I

CONCEPT OF THE 6600 CENTRAL PROCESSOR



A GENERAL AND OVERALL 6600 COMPUTER BLOCK DIAGRAM

CHAPTER I

CONCEPT OF THE 6600 CENTRAL PROCESSOR

INTRODUCTION

The CONTROL DATA® 6600 Computer System, through use of high-speed transistor logic and a design philosophy based on concurrent (or parallel) processing, is today recognized as the world's fastest and most powerful computer. The rapid throughput achieved by the 6600 system can be attributed in part to the concurrency that exists in several areas of the Central Processor.

MEMORY BANK PHASING

The Central Memory is divided into memory banks, each of which contains $4096_{(10)}$ 60-bit central processor words. A 131K central memory is composed of 32 such banks; a 65K memory has 16 banks. Since each bank has its own circuitry for the X & Y drive lines, inhibit lines, sense lines and memory cycle timing, each is capable of operating independently. This, in turn, permits memory cycles to be phased (overlapped) by 100 nanoseconds, to effectively reduce minimum access time to 100 nsec (e.g., a memory cycle is one microsecond in duration, but ten may be initiated each usec as long as they are to different banks). The bank phasing scheme, in addition to a memory cycle which is in itself extremely fast, eliminates a great portion of the memory waiting time that is inherent in the majority of computers.

INSTRUCTION STACK

A group of flip-flop registers referred to as the Instruction Stack is provided in the 6600 for the purpose of holding an iterative sequence of instructions (a program loop). The Stack can hold a loop containing up to 27 instructions (up to 4 instructions per word) which may then be executed without the need for instruction word memory references (RNIs).* Initially, the eight stack registers (I registers) are filled by reading instruction words from central memory. As each word is read into and executed from I0 (See Figure 1-1), the preceding words move up in the stack and a new word is entered into the first I register. When the stack is filled, the movement of instruction words causes the top word (in

---

* Although 27 instructions may at first appear to limit the programmer's capability, it should be considered that the 6600 is designed primarily as a scientific machine. Consequently, a good many programs will be of a mathematical nature (i.e., matrix analysis). Also, each instruction can designate two source operands and one result destination. When viewed in this light, 27 instructions are, in most cases, more than adequate.

®Registered trademark of Control Data Corporation.

17) to be discarded. When instructions are being executed in the stack (looping), no movement occurs and the stack information remains static. In this manner, the necessity of fetching each instruction from memory is eliminated during short loops. The memory access time savings should be obvious.

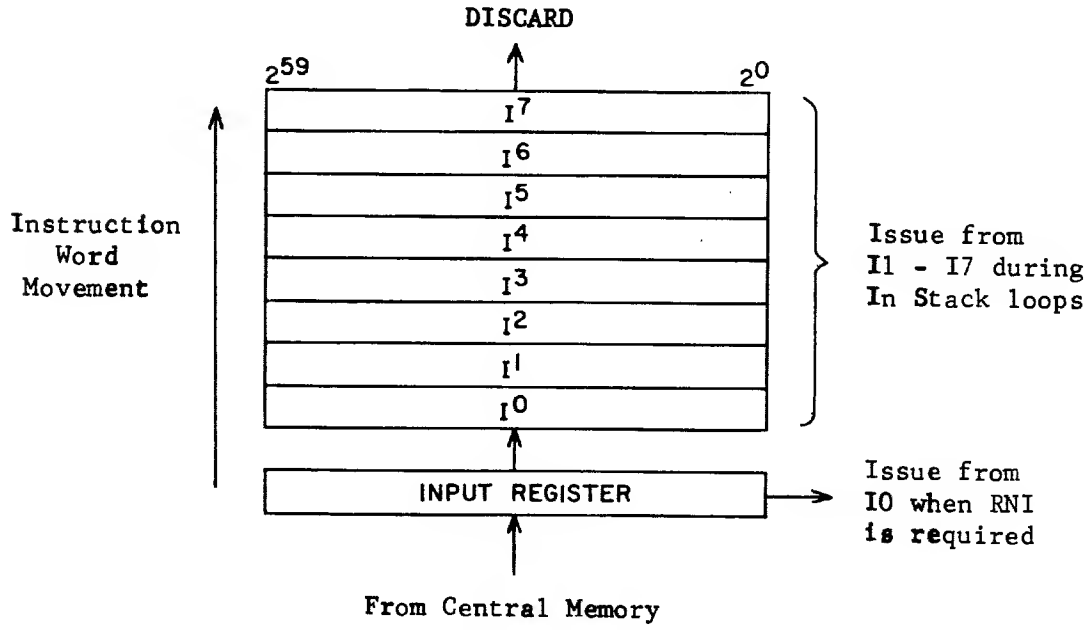


Figure 1-1

Operating Registers

Another property of the Central Processor decreases the number of memory references required to read and store operands. Twenty-four operating registers provide a flip-flop storage facility for 60-bit operands and 18-bit addresses and indexing values. Eight 60-bit registers (designated X0 - X7) provide for the storage of integer and floating point values in a 60-bit format. Eight 18-bit registers (designated A0 - A7) provide storage for central memory addresses of operands which are read or stored in memory. Eight 18-bit registers (designated B0 - B7) provide for storage of indexing values, used for modification of addresses and operands. Since most central processor instructions can designate two source operands (taken from X, B or A registers) and one result destination (X, B or A register), considerable operand manipulation can take place by use of the operating registers, thereby further decreasing the number of memory accesses needed. Proper use of the instruction stack and operating registers makes possible, execution of program loops which require no memory references -- for instructions, operands or storage of results.

Functional Units

Another area of concurrency in the 6600 Central Processor is that of parallel arithmetic (functional) units. Ten logically independent functional units are provided to allow several instructions to be in various stages of execution at the same time. The following list describes the functional units and their corresponding cycle times:

	UNIT	TIME (nanoseconds)
1)	ADD (floating)	400
2)	MULTIPLY 1 (floating)	1000
3)	MULTIPLY 2 (floating)	1000
4)	DIVIDE (floating)	2900
5)	BOOLEAN (logical)	300
6)	LONG ADD (integer)	300
7)	SHIFT	300 - 400
8)	INCREMENT 1 (indexing)	300
9)	INCREMENT 2 (indexing)	300
10)	BRANCH (branch instructions)	800 - 1400

Each unit is assigned a group of instructions which it, and only it, processes. For example, the ADD unit processes all single precision, double precision, rounded and unrounded floating point add opcodes. The SHIFT unit handles opcodes that require shifting: left and right shifts, normalize operations, packing, unpacking, etc.

Separate functional units eliminate the necessity for sequential execution of program steps, a property which is inherent in most present-day computers. Instead, unrelated instructions may be processed out of sequence, causing a considerable decrease in the over-all execution time of a program. Of course, if a source operand for one unit is the result operand of another, the first unit must wait until the second completes its calculation and returns the result. Also, if two division steps are needed in sequence, the second must wait until the first completes, since only one divide unit exists. On the other hand, two multiply operations may take place at the same time because two multiply units are provided. The point to be stressed is that in most operational programs the instructions need not be executed in sequence. Instead, the majority of problems are composed of a series of smaller steps which are only indirectly related. The following programming comparison should illustrate this point.

The problem that follows is solved first by using a sequential computer and secondly, by using the 6600 with its functional units. Individual instruction execution times are assumed to be the same in both machines. Also, both have the capability of reading two.

source operands and returning one result by use of operating registers (X, B and A).

THE PROBLEM:

$$\left(\frac{A + B}{C}\right) \cdot (A^2 + B^2 + C)$$

THE OPERATING REGISTER CONTENTS: (where () means "the contents of")

(X1) = the value, A
(X2) = the value, B
(X3) = the value, C

THE PROBLEM THUS BECOMES:

$$\left[\frac{(X1) + (X2)}{(X3)}\right] \cdot \left[(X1)^2 + (X2)^2 + (X3)\right]$$

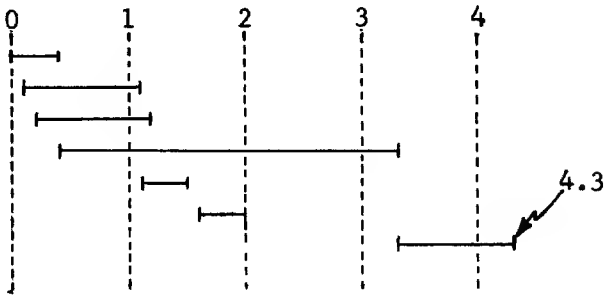
USING THE SEQUENTIAL COMPUTER: (where → means "replaces")

Instructions	Time (nanoseconds)
1. (X1) + (X2) → (X4)	400
2. (X4) / (X3) → (X5)	2900
3. (X1) * (X1) → (X6)	1000
4. (X2) * (X2) → (X7)	1000
5. (X6) + (X3) → (X0)	400
6. (X0) + (X7) → (X7)	400
7. (X5) * (X7) → (X6)	1000
TOTAL TIME =	7100

Since the instructions must be executed in sequence, the total execution time is the sum of the individual execution times, or 7.1 microseconds.

USING THE 6600:

1. (X1) + (X2) → (X4)
2. (X1) * (X1) → (X6)
3. (X2) * (X2) → (X7)
4. (X4) / (X3) → (X5)
5. (X6) + (X3) → (X0)
6. (X0) + (X7) → (X7)
7. (X5) * (X7) → (X6)



(NOTE: Time is shown in microseconds)

Using parallel functional units, the program execution time is only 4.3 microseconds, a reduction of approximately 40%.

Although the same saving will not occur in all programs, the example illustrates that, through efficient programing, a considerable decrease in execution time occurs. Even when a program is not optimized, a time saving will be realized. Details of time implications from the preceding chart are considered in later topics.

Summary

Several unique features are incorporated in the design of the 6600 central processor, including: 1) thirty-two (or sixteen) 4K, phased memory banks, 2) an instruction stack containing eight 60-bit registers, 3) twenty-four operating registers and 4) ten independent functional units. These provisions work in conjunction with each other to provide extremely rapid program execution times. Whenever parallel processing capabilities are provided in a computer, control circuitry is required to ensure that all features work together (without calamity) to produce a high-speed processing system.

BLOCK DIAGRAM ANALYSIS

CENTRAL MEMORY ADDRESS CONTROL

References to Central Memory can be initiated from various sources in the 6600. Peripheral Processors make central memory references during the central read, write and exchange jump instructions. The Central Processor uses central memory to fetch instruction words or to read and store operands. An orderly means for handling these memory requests and distributing the associated data must be utilized. This is complicated by the fact that the 6600 memory banks are phased to allow several memory cycles to be in progress at any one time. Therefore, it is very possible that a memory reference request be made to a bank that is already busy processing a memory cycle, so that the address must be saved and then re-issued. It is also conceivable that two requests occur simultaneously, requiring that a decision be made regarding which address will be issued first. Not only must the address be manipulated methodically, but the source or destination of the data associated with each address must be "remembered" by the control logic. These functions are accomplished for the most part, by the Central Memory Control logic, more often referred to as the Stunt Box.

Analysis of the Stunt Box takes place in the following sequence:

1) Hopper

2) Priority Network

3) Tag Generation and Distribution

HOPPER

The Hopper is a mechanism used to save conflicting addresses so they may be re-issued to the memory banks repeatedly, if necessary, until accepted and processed. Along with addresses, the Hopper saves gating information used to enable the data corresponding to each address through the memory Data Distribution logic to or from memory.

Physically, the Hopper is four flip-flop registers (designated M1, M2, M3 and M4) each of which stores an 18-bit address, 6-bit tag and a Full bit (except M2, which has no Full bit). Refer to Figure 1-2. The registers are connected to each other in such a manner as to allow information to circulate through each of the registers (the concept is similar to the Peripheral Processor barrel). A 75 nanosecond time interval exists between each register and produces a total re-circulation time of 300 nsec. For example, an address entered into M1 at time 00 enters M4 at t75, M3 at t150, M2 at t225 and (if it must be re-issued) re-enters M1 at t300.

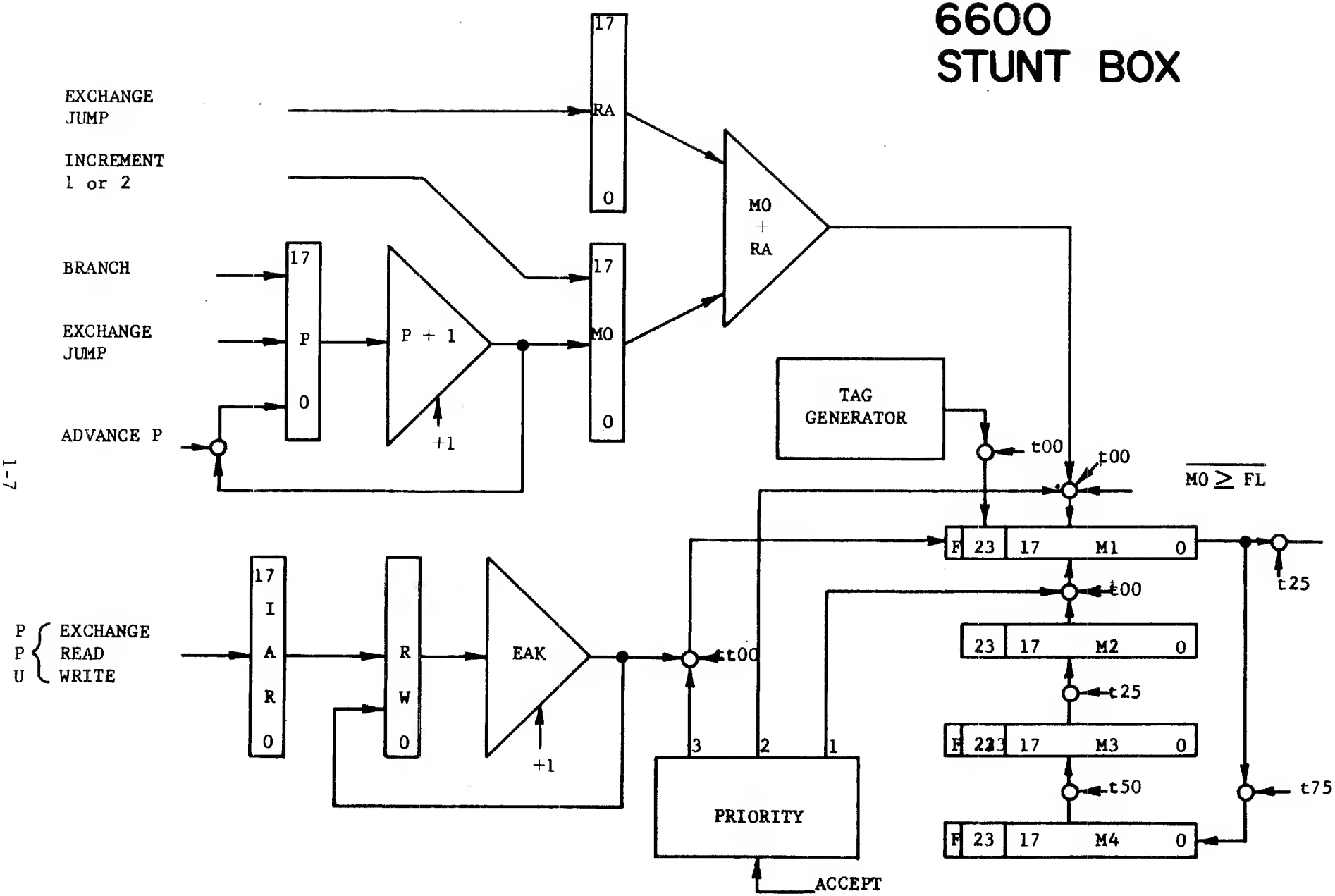

6600
STUNT BOX
EXCHANGE JUMP
INCREMENT 1 or 2
BRANCH
EXCHANGE JUMP
ADVANCE P
17
RA
0
17
P
0
P + 1
+1
17
MO
0
MO + RA
TAG GENERATOR
t00
t00
MO ≥ FL
F 23 17 M1 0
t25
t00
23 17 M2 0
t25
F 23 17 M3 0
t50
t75
F 23 17 M4 0
P EXCHANGE
P READ
U WRITE
17 I A R 0
R W 0
EAK
+1
t00
3
2
1
PRIORITY
ACCEPT

Figure 1-2

In actuality, only 17 of the 18 available address bits are used in a 131K central memory. The 18th bit is found throughout the memory circuitry, but is never utilized. The 6-bit tag is generated when an address is first entered into the hopper (specifically, M1) and it contains all the information necessary to properly distribute the associated data. Tag generation and distribution is almost a subject in itself and is treated separately later in this section. The Full bits found in M1, M3 and M4 indicate that a meaningful address and tag are contained in the respective register. It is set when an address and tag are entered into M1. There is no full bit in M2, since the Accept signal (explained below) serves a similar purpose.

Approximately 50 nsec after entering an address into M1, the address is automatically sent to the memory banks, where the lower 5 bits are examined to select one of the 32 banks. If the desired bank is not in use ($\overline{\text{busy}}$), an Accept signal is returned to the Stunt Box to indicate that no conflict exists and the memory cycle has been initiated. The address saved in the Hopper is then discarded. If the desired bank is busy, an Accept will not be returned. Its absence causes the associated address to be re-entered into M1 (from M2) and subsequently, reissued to the memory banks. The cycle will recur every 300 nsec until the address is accepted.

Figure 1-3, a timing diagram, verifies that the Accept is returned to the Stunt Box in time to disable (or if not returned, enable) the transfer of M2 to M1. If an Accept is generated, it will be received on Chassis 5 about 175 nsec after entering the associated address into M1 (t175). This allows 125 nsec of logic delay time before the Accept is used to disable the M2 ⟶ M1 transfer (t300). Since M2 was not transferred to M1, the following transfer of M3 to M2 will destroy (write over) the contents of M2.

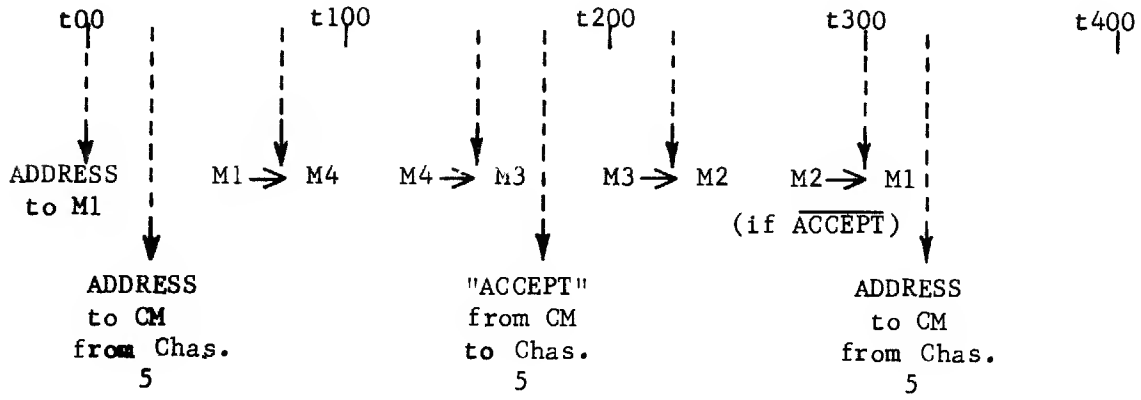


Figure 1-3.

PRIORITY NETWORK

Because more than one source exists for addresses entered into M1 (See Figure 1-2) a Stunt Box Priority Network is necessary to ensure organized handling of simultaneous memory requests. Each address source has a fixed priority, as follows:

First	Hopper (M2 $\longrightarrow$ M1)
Second	Central Processor (M0 $\longrightarrow$ M1)
Third	Peripheral Processor (ERW $\longrightarrow$ M1)

HOPPER PRIORITY

In only one circumstance is re-entry of an address to M1 required: when an address has been sent to the memory banks and was not accepted due to a bank conflict. Non-acceptance of an address is indicated by not receiving an Accept from the memory banks 175 nsec. after issuing an address. Before enabling an M2 $\longrightarrow$ M1 transfer it should also be determined that M2 contains a meaningful address. This is indicated by the presence of a Full bit. Since M2 does not contain a Full bit, the M3 Full bit is checked. (It is time delayed to ensure that M3 has been transferred to M2 before the check is made.) Thus, two conditions must be met to grant first priority:

(M3 Full) $\qquad$ ($\overline{\text{Accept}}$)

CENTRAL PRIORITY

Two sub-priorities exist under Central Priority because central processor memory references may be originated in two independent operations

1) Instruction word fetching (RNI's)

2) Reading and storing operands

In the first case, the address is obtained from the P register and in the second, from one of the two Increment Functional Units. If requests from both sources occur simultaneously, the operand address is entered first, then the instruction address. In either case, the address is entered into an 18-bit register, M0 (Figure 1-2). At the same time a control flip-flop called "Enter Central" is set and indicates that an address is in M0 waiting for entry into M1. (In a sense, the Enter Central flip-flop requests priority #2). Thus, one condition required for priority #2 is that Enter Central is set to indicate that an address is in M0 waiting for entry into M1.

A second condition needed for central priority is that priority #1 does <u>not</u> exist (i.e., the address in M2 was accepted or M2 does not contain a meaningful address).

A special circumstance arises which also must be considered in granting central priority. This is the case when read and store requests are made to the same memory address. This might occur when an instruction word modification is made followed by an RNI request for the modified word. If the two addresses enter the hopper in sequence (store location X, then read location X) storing before reading cannot be guaranteed because a bank conflict may exist with the store address. The operation (read or store) that is performed first depends strictly on when the bank goes BUSY. Whichever address is sent to the banks first (after BUSY) will be accepted and will cause a conflict for the second reference to the same location. Thus, it would be possible, in the above instance, to read the unmodified instruction word when actually, the modified word was desired. The reverse situation might also occur, wherein a location was to be read before modification.

To resolve the above cases, additional logic is required in the priority #2 circuitry which prevents a Central Read address from being entered into the Hopper if any (Peripheral or Central) Write address is in the Hopper. Also, if a Central Write is attempted, no Central Read address may be in the Hopper. (Prevention of a Central Write and Peripheral Read out of sequence is a software responsibility.)

The fourth, and final condition needed for Central Priority, is that the address being referenced must not be out of the bounds for this particular program. Memory bounds for a program are defined upon initialization of the routine (EXCHANGE JUMP) by the RA (Reference Address) and FL (Field Length) values. RA specifies the lower bound and RA + FL -1, the upper bound. Each central memory reference adds to the value RA, the content of P (for RNI's) or the Increment I or II address (for operand references). Thus, the address being referenced (P, Incr. I or Incr. II) is said to be the "relative" address. The absolute CM address is the sum of the relative address and the content of RA. The relative address is always entered into MO. A special Adder adds MO to RA and yields the absolute address. Another circuit compares the content of MO with the content of FL. If MO$\geq$FL, the desired reference is "Out of Bounds", and the memory reference will not take place because Central Priority will not be granted. Thus, the condition MO$<$FL is also a condition required for granting Priority 2.

The following Boolean formula summarizes the conditions required for granting Central Priority:

$$(\text{Enter Central})\ \overline{(\text{Priority 1})}\ (\text{MO}<\text{FL})\ (\text{Attempt Read})$$

$$\overline{(\text{Write in Hopper})} + (\text{Attempt Write})\ \overline{(\text{Central Read in Hopper})}$$

PERIPHERAL PRIORITY

Peripheral priority for CM references is granted only if neither Hopper nor Central priority exists and there is a peripheral processor request for a CM access. Since only one PPU request can occur at a time, no sub-priorities are required.

The PPU's request CM references in three situations:

1) Read central memory

2) Write central memory

3) Exchange jump.

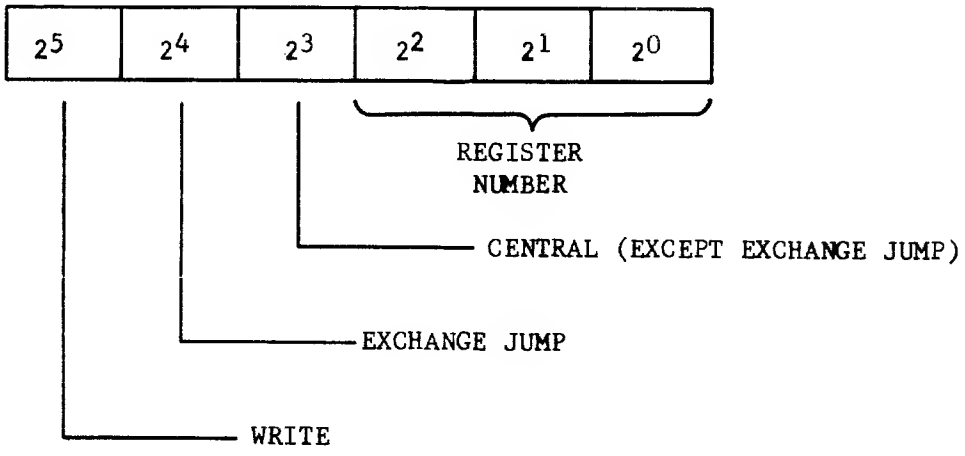
In all three cases, a PPU will send an 18-bit address to the Input Address Register (IAR) of the Stunt Box (See Figure 1-2). To specify the type of reference being requested, a Read , Write or Exchange pulse accompanies the address. These are used as a control function, to properly gate information to and from CM (See Hopper Tag discussion which follows). The presence of one of the three control pulses results in the Request for Priority #3. Thus, the Boolean expression for Peripheral Priority is as follows:

$$(\text{PPU Read} + \text{Write} + \text{Exchange})\ \overline{(\text{Priority 1})}\ \overline{(\text{Priority 2})}$$

During peripheral processor read and write operations in central memory, a new address is sent to the IAR for every memory reference desired. For exchange jumps, only the starting address of the exchange jump package (in CM) is sent. It is the responsibility of the central processor to advance this address automatically in order to exchange the required information. This is accomplished by the Exchange Address Counter (EAK) which is utilized only during exchange jumps. It increments the exchange address for each of the 16 locations referenced.

HOPPER TAG GENERATION AND DISTRIBUTION

As previously mentioned, when an address is entered into M1, a 6-bit tag is also entered. It is used to properly gate data into and out of Central Memory. The tag bit positions are named as follows:



The bit (2^5) will be set any time the associated address is that of information to be stored (written) into Central Memory.

The bit (2^4) is set only during exchange jumps to indicate that the associated data is to be exchanged with registers in the CPU.

The bit (2^3) is set any time a memory reference is initiated by the Central Processor (Priority 2) and allows information to be gated to or from the CPU, as opposed to a PPU. During exchange jumps, the bit is set to indicate that an X register is to be exchanged, or cleared to indicate that A, B and Control Registers (P, RA, FL, etc.) are to be exchanged.

The bits (2^0 - 2^2) indicate (when applicable) which X, B, or A register number is to be stored, read into, or exchanged. Table 1-1 lists all legal tag numbers (in octal) and their meaning. Decoding circuitry exists only for those tags listed. Any other bit combination will either not be decoded, or will be decoded as one of the legal tags.

TABLE 1-1. HOPPER TAGS

00	Peripheral Read	63	Exchange EM, A3, B3
10	CP RNI	64	Exchange RA(ecs) A4,B4
11	CP Read → X1	65	Exchange FL(ecs) A5,B5
12	CP Read → X2	66	Exchange A6,B6
13	CP Read → X3	67	Exchange A7,B7
14	CP Read → X4	70	Exchange X0
15	CP Read → X5	71	Exchange X1
40	Peripheral Write	72	Exchange X2
50	Return Jump + Error Stop	73	Exchange X3
56	CP Write X6	74	Exchange X4
57	CP Write X7	75	Exchange X5
60	Exchange P, A0	76	Exchange X6
61	Exchange RA(cm) A1,B1	77	Exchange X7
62	Exchange FL(cm) A2,B2		

A tag = 00 indicates a Peripheral Read address since all bits equal zero. This is interpreted as meaning:

($\overline{\text{WRITE}}$) ($\overline{\text{EXCHANGE}}$) ($\overline{\text{CENTRAL}}$) or Peripheral Read.

In this case the register bits (2^0 - 2^2) have no meaning and are not translated.

A tag = 10 indicates a Central Read Next Instruction (RNI) since the Central bit (2^3) is set and all other bits are cleared. Since a Central Read of Memory to X0 is not possible, the clear state of bits 2^0 - 2^2 in this case indicate that an instruction word is to be read from Memory.

A tag = 11 indicates that a Central Read to X1 is to be performed. Bits 2^0 - 2^2 in this case indicate the X register number. Tags 12-15 are also Central Reads to X registers, but to X2 - X5, respectively.

A tag = 40 indicates a Peripheral Write operation, since the write bit (2^5) is set and the Central and Exchange bits (2^3 & 2^4) are both cleared.

A tag = 50 indicates a Central Return Jump or Error Mode Stop memory reference. Bits 2^0 - 2^2 are meaningless in this case since a Central Write (tag = 5X) of X0 is not possible. Since storage of information in central memory is required in the above cases, the 50 tag is reserved for this purpose.

Tags = 56 & 57 are generated when storage of X6 or X7, respectively, is desired. Bits 2^0 - 2^2 again indicate the register number.

Tags 60 - 77 are all generated during an Exchange Jump operation. Bit 2^3 = 0 indicates that A, B or Control registers are to be exchanged. Bit 2^3 = 1 indicates that an X register is to be exchanged. Bits 2^0 - 2^2 specify the operating register number (i.e.,X, B or A) or the control register (i.e. P, RA, FL, or EM) to be exchanged. Note that these are the only cases when bit 2^4 (the Exchange bit) is set.

After a memory reference is initiated, the associated tag is decoded and will enable the gating of the desired information into and/or out of Central Memory, to or from the desired location (XBA registers, control registers, read or write pyramids, etc.).

CENTRAL MEMORY

The 6600 Central Memory is composed of 60-bit words located in 16 or 32 memory banks each of which contains 4K words. This results in 65K or 131K memory sizes, respectively. In either case, 4 banks are contained on a chassis.

Selections of bank and chassis are made by decoding the lower 4 (for 65K memories) or 5 bits (for 131K memories) of the address. For example, in 131K system,

bits 2^0 and 2^1 select one of 4 banks on a chassis, while bits 2^2, 2^3 & 2^4 select one of 8 chassis. The address is sent from Chassis 5 to all memory chassis of a system and all chassis decode the lower bits of the address. Only one chassis will recognize its bit configuration (2^2 - 2^4). By decoding bits 2^0 and 2^1, the bank selection is made. If the selected bank is free (i.e.,a memory cycle is not already in progress) the Accept signal is returned to the stunt box and a Go signal is sent to the selected Storage Sequence Control circuit (SSC). The SSC is a simple flip-flop timing chain which generates the read/write memory cycle.

The selected address within the selected bank is determined by decoding the remaining 12-bits of the 17-bit address (16 bits for a 65K system). While a memory cycle is in progress, the bank busy signal (bank not free) disables initiation of other memory cycles within that bank. It also disables the return of the Accept signal to Chassis 5, which causes the address to be retried at the 300 nsec stunt box rate.

The information being read from or stored into central memory is gated by a circuit called the Read/Write distributor. It, in essence, distributes information to and from the 4 Chassis connected to central memory as shown in Figure 1-4.

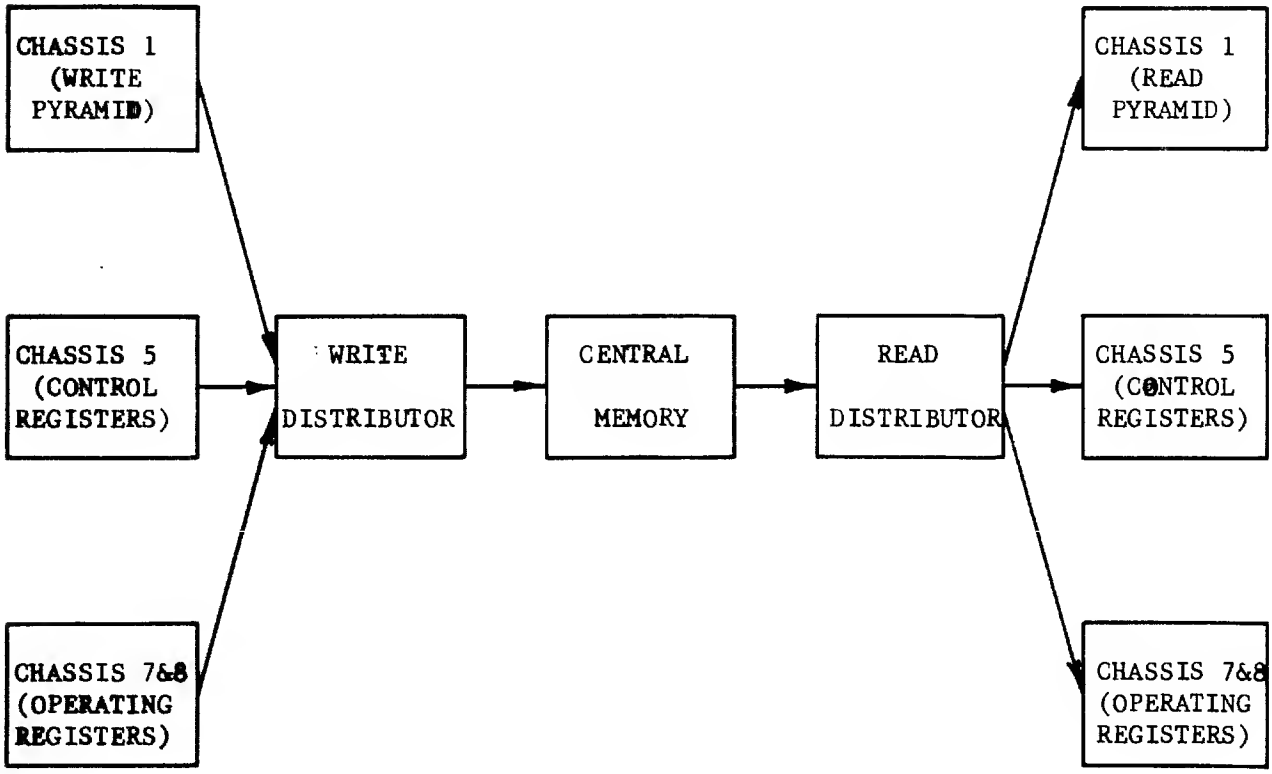


Figure 1-4.

Determination of which information is to be gated is made by decoding of the Hopper tag associated with each address and ANDing the decoded signal with the Accept for that address.

Information is gated from the Write Pyramid on Chassis 1 to CM via the write distributor during PPU central memory write operations (tag = 40).

Information is gated from Chassis 5 during return jumps and Error Mode Stops (tag = 50).

Information is gated from Chassis 7 and 8 during Exchange Jumps (tags = 60 - 77) and central processor store operand operations (tags = 56 or 57).

Information is sent from C.M. via the Read distributor to Chassis 1 during PPU central memory read operations (tag = 00).

Information is sent to Chassis 5 during Exchange Jumps (tags = 60 - 65) and RNI (tag = 10) operations.

Information is sent to Chassis 7 and 8 during Exchange Jumps (tags = 60 - 77) and central processor operand read (tags = 11 - 15) operations.

It is re-emphasized that in all cases of gating the Read/Write distributor the Accept signal is necessary. This ensures that the information desired is properly timed for entry to or exit from memory.

INSTRUCTION CONTROL

Instructions in the Central processor are executed from the Instruction Stack (shown in Figure 1-1). Each 60 bit Stack register can contain up to four instructions, since the Central processor employs both a 15 bit and a 30 bit instruction format, and as few as two. The responsibility of Instruction Control is to determine that a 60 bit Instruction word has become available to the stack, sort out the 15 or 30 bit instructions within that word, and then deliver the instructions to Reservation Control so they can be executed.

Initially all instruction words (a 60 bit memory word fetched by an RNI request) move into the bottom rank of the stack (I0) when Instruction Control receives the RNI tag (TAG = 10_8 and accept) from the Stunt Box. The RNI tag also signals Instruction Control to begin the process of sorting instructions within that word and transfering them to Reservation Control. The total process is called Instruction Issue, and the sorting of instructions is referred to as Parcelling.

INSTRUCTION ISSUE

Instructions can be issued from any rank of the stack, however if we were to assume an initial condition such as at the end of an Exchange Jump sequence we would see issue beginning with the upper instruction in I0. Once a program is in execution, program control can be transferred to some higher rank of the stack by a Branch instruction. This situation forces Instruction control to keep track of which rank of the stack the Program address is currently indicating. The "Locator" (L) register and counter perform this function. Control of the L count would be very similar to control of the Program Address. However, L refers to a particular rank of the stack so it would only vary between 0_8 through 7_8. Example: L count = 0_8 indicates program control is currently in I0. L count = 7_8 indicates program control in I7. An initial Master Clear would set the L count to 0_8, so we can see that Instruction Issue would start from the bottom rank of the stack.

NOTE: The L register contains the complement of the L count.

Now that we have selected a particular rank of the stack, we must concern ourselves with sorting out the instructions within that rank. In other words we must parcel the instructions from the selected rank of the stack. Each rank is considered to have four overlapping 30-bit Parcels.

These are: parcel 0 - bits 30 through 59
parcel 1 - bits 15 through 44
parcel 2 - bits 00 through 29
parcel 3 - bits 44 through 14 (end around)

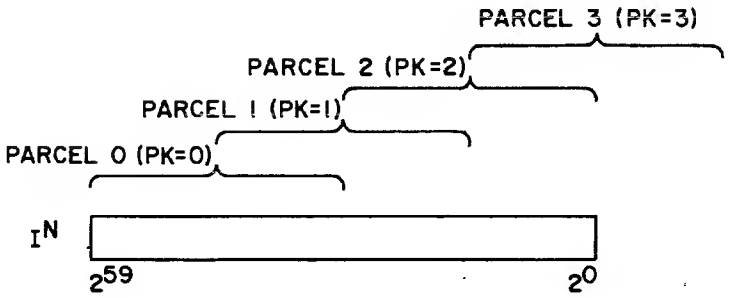
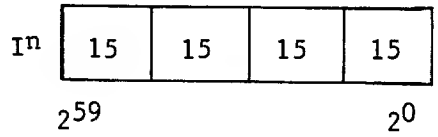


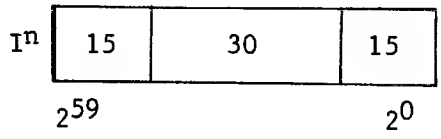
Figure 1-5 Instruction Word Parcels

By examining each word with the 30 bit parcels we guarantee the detection of any possible combination of instructions within that word. If the instruction contained within the parcel happens to be a 15 bit instruction the lower 15 bits of the parcel are discarded and the next sequential parcel is extracted. However, if a 30 bit instruction is encountered, the entire parcel would be used and the next sequential parcel would be skipped.

Example:



Here each parcel would be extracted in sequence with the lower 15 bits of each parcel being discarded.



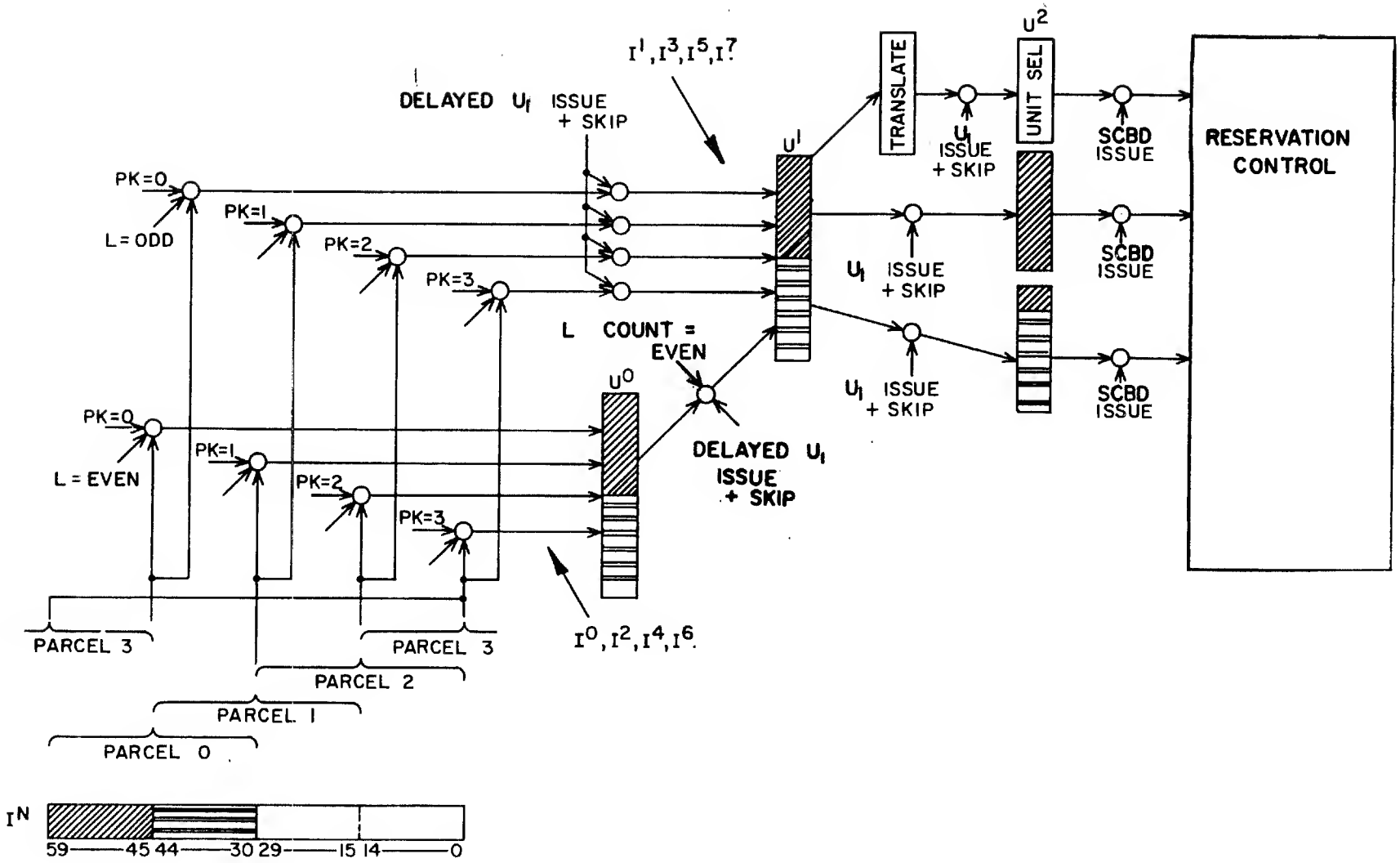


Figure 1-6 Parcel Counter

In this example parcel 0 and 1 would be extracted in sequence all 30 bits of parcel 1 would be used, so parcel 2 would be skipped.

Instruction Control uses a two bit Parcel Counter (PK) to keep track of the current parcel and will generally advance the parcel counter after each parcel is extracted. An initial PK = 0 condition would be set by Master Clear.

Initial Instruction issue begins with the L count = 0 (I0), PK = 0 (bits 30 through 59), and, as previously mentioned, the first Instruction word after an Exchange Jump comes from Central Memory to I0 as a result of the RNI tag. The tag also starts the Issue operation, so the rest of our analysis can now be concerned with moving the instructions to Reservation Control. Figure 1-B shows the path each parcel will take.

Issue Control, generates two types of issue pulses. These are:

U1 issue - A pulse that gates the selected parcel to the U1 and U2 instruction registers and advances PK. This pulse occurs at a minor cycle rate during the issue sequence.

Scoreboard Issue - A pulse that gates the parcel from the U2 instruction register to Reservation Control. This issue can also occur at a minor cycle rate.

With eight different L counts and four different Parcel counts, it is easy to see that 32_{10} different parcels must move through the U1 and U2 registers. Sixteen of these parcels(PK = 0, 1, 2, 3 L count = 0, 2, 4, 6) move from the even-numbered ranks of the stack to U0 before U1 issue would move then to U1. It is not necessary to have an issue pulse to move parcels to U0, so we would see the selected parcel from the selected even rank of the stack move into U0 automatically. In our initial case Parcel 0 of I0 would be the first parcel extracted to U0 and the first U1 issue pulse would move the parcel to U1.

Notice that at this time there have not been meaningful parcels in U1 or U2, so as far as the U2 register is concerned it receives "Trash" on the first U1 issue. Also, no scoreboard issue should be generated until after the first meaningful parcel has moved into U2. The PK being advanced to 1 by U1 issue would cause parcel 1 of I0 to be extracted to U0, so on the next U1 issue parcel 1 would move to U1, parcel 0 would enter U2, and PK would advance to 2.

One more U1 issue would move parcel 2 to U1, parcel 1 from U1 to U2, so at this time the first Scoreboard issue must occur to issue parcel 0 to Reservation Control. From this point both U1 and Scoreboard issue can continue at a minor cycle rate until parcel 3 is issued to Reservation Control (three more issue pulses).

All of the possible instructions in I0 have now been put into

execution, and issue must stop until the next 60-bit Instruction word becomes available from Central Memory. This is called a "Pause".

The Pause could be quite lengthy if Instruction Control had not had the foresight to request another RNI from Central Memory. This request is made any time L count = 0, PK = 0, and U1 issue. It is easy to see why the request is made under those conditions, once it is realized that I0 is the bottom rank of the stack and after issuing from I0 there wouldn't be any place to go for the next instruction. There is one other operation that comes into play at this time, and it is the process of moving the current instruction words of the stack up to make room for the next instruction word from Central Memory (Inching).

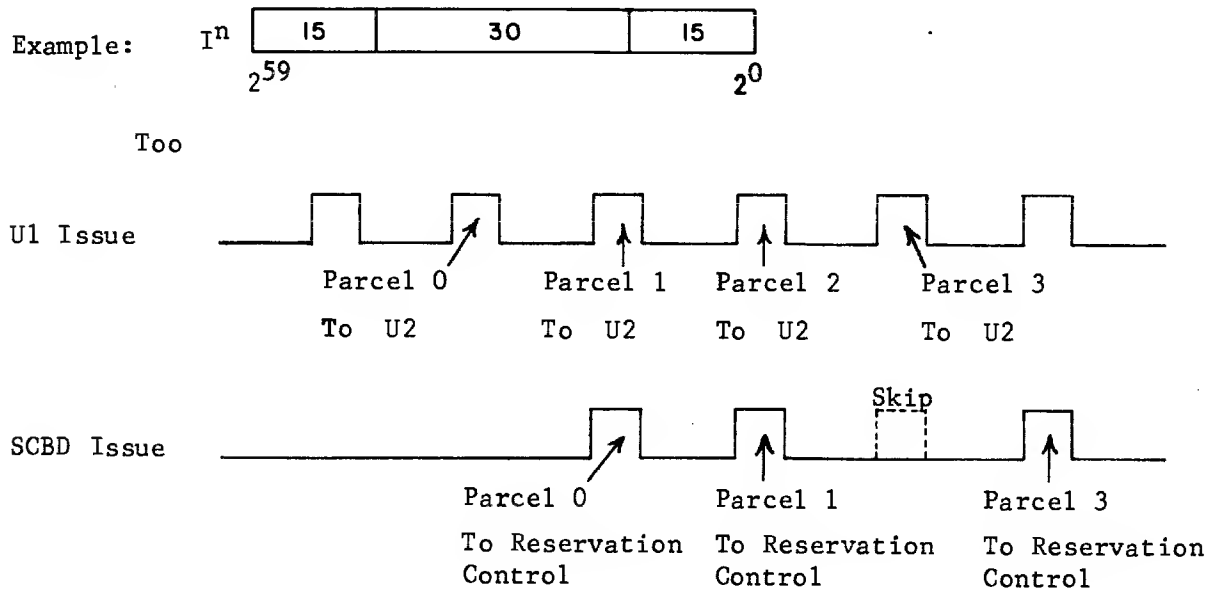
Inch is also started when L count = 0, PK = 0, and U1 issue. During the Inch process each rank of the stack is shifted up starting with an I6 to I7 transfer and continuing with I5 to I6, I4 to I5, I3 to I4, I2 to I3, I1 to I2, and I0 to I1 in that sequence.

Four minor cycles (400 nsecs) are necessary to complete the Inch, and it becomes necessary to advance the L count to 1, since the last Inch transfer moves the current Instruction word from I0 (L count = 0) to I1 (L count = 1). An important point to realize here is the I6 to I7 transfer destroys the Instruction word that was in I7. Consequently, a program loop that is to be executed within the Stack must fit in the stack between I1 and I7. A quick examination of the stack reveals a maximum in Stack Program length of 27_{10} instructions.

I7	15	15	15	15
I6	15	15	15	15
I5	15	15	15	15
I4	15	15	15	15
I3	15	15	15	15
I2	15	15	15	15
I1	15	15	30 Bit Branch	
I0				

Figure 1-7 Maximum In Stack Loop

The analysis of Instruction Issue to this point has assumed straight line program execution with no complications. There are, however, many special situations which may be encountered. Whenever a 30-bit Instruction is encountered in a parcel, Instruction Control must cause the next sequential parcel to be skipped. The skipping is accomplished merely by not generating a Scoreboard issue when the unwanted parcel is in U2.



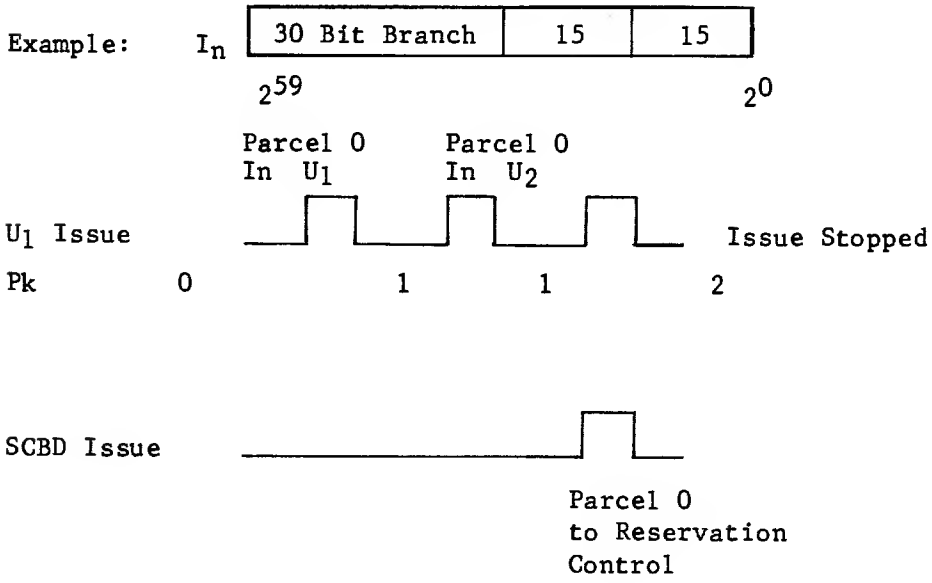
There is also the possibility that Issue may have to be stopped if either the Functional Unit or the Result register (required by the parcel) are Busy. This is accomplished by Instruction control translating the parcel when it is in U_1 and setting a Unit Request FF and Result Register Select FF when the parcel enters U2. Each Unit Request FF interrogates the corresponding Unit Busy FF in Reservation Control, and, if the Unit is busy, a signal is generated which blocks all Issue pulses un il the Unit becomes not busy.

A similar operation occurs with the Result register except the Result Register Select FF must be ANDed with a translation for the "i" portion of the Parcel to determine which portion of the Reservation List should be examined for a Reservation. The Reservation List (XBA) is where all Result register reservations are held by Reservation Control.

Probably the most involved operation in Instruction Control occurs when a Branch Instruction is encountered, and, even though the Branch Unit will execute the instruction, Issue Control must set itself up for proper operation. The reason for the complication of course, is that a Branch Instruction can do one of the following three:

Loop - a conditional Branch, condition met, and In Stack.
Jump - an unconditional Branch or a conditional branch, condition met, and not In Stack.
No Branch - a conditional Branch, condition not met.

Instruction Control will set itself up for the No Branch Condition by setting the Parcel Count equal to the parcel count of the next parcel to be issued, and then stopping Issue after issuing the Branch.



By controlling the Parcel count in this manner it is possible to restart issue the same as if it were starting after an RNI. Once the Branch has been issued, the Branch Unit makes a series of tests to determine whether the branch is to an instruction word already in the Stack.

The test results are only enabled on the Conditional Branch instructions 03_8 through 07_8 and enable these instructions to Loop. The first test is made by subtracting the current Program Address (P) from the Jump Address (R), and if the difference (T) is ± 7 or less the Branch may be In Stack.

However, a further test must be made to see if a jump of T places can be made relative to the current position in the Stack which is reflected by L. This is the L-T test and, if there is not an end-around borrow from the test, the branch still may be In Stack.

If R-P gave a positive result, the jump was forward and the L-T test being made successfully would say In Stack, but if R-P was negative, the branch was backward and a further test must be made to see if there is a usable instruction in the rank of the stack to which the jump is being made. Conveniently, the result of the L-T test would be the new L setting if the Branch is to be made, and this is subtracted from the Stack Depth Counter register (D). The D-(L-T) test is only necessary on the backward jumps (R-P negative) and, if it is successfully made, the branch would be In Stack. The Branch unit uses the Long Add unit to make the Branch Condition test for the 03_8 instructions, and the Increment units to test the 04_8 through 07_8 instructions. If the condition is met, a Go Branch signal is generated. If the R-P, L-T, D-(L-T) have all been successful, a Loop Proceed is generated.

If Go Branch occurs and R-P, L-T, and D-(L-7) were not successful, a Jump is generated. If Go Branch does not occur, a No Branch Proceed is generated. On a Loop Proceed, the Jump Address (R) is transferred to P, the result of the L-T test is gated to L, the Parcel Count is set to 0_8, and Issue is restarted. On the Jump an R to P transfer is also accomplished, but L is set to 7_8 (L count = 0_8), the Parcel count is cleared, and an RNI request is made to the Stunt Box. (Issue would restart as a result of the RNI.) The No Branch Proceed merely restarts Issue, since this is why Instruction Control has been set up. There are many special cases that affect Instruction Control during branch instructions, but these will be covered in Section 4 along with a more detailed explanation of the other Instruction Control operations.

RESERVATION CONTROL
(SCOREBOARD)

The need for reservation control logic in the 6600 Central Processor arises due to the parallel processing concept of the CPU. This capability necessitates an orderly means of utilizing the functional units, operating registers and memory circuitry, since it is possible that several instructions require the same functional unit, operating register, etc. The scoreboard , then, makes the required reservations of each instruction and provides a means for the orderly handling of conflicts which may occur between instructions.

Conflicts are categorized into three groups - first, second and third order. The types of conflicts are defined as follows:

1) FIRST ORDER: A conflict between two instructions that require the same functional unit or the same result registers.

EXAMPLE 1: Functional Unit Conflict

F X 6 = X1 ⊕ X2
F X 5 = X3 ⊕ X4

Both instructions need the Floating Add functional unit for their calculation. Since only one such unit exists, the second instruction must wait until the first is finished, before it can be executed. Note that if two Multiply instructions are coded in sequence, no functional unit conflict occurs since two multiply units are provided.

EXAMPLE 2: Result Register Conflict

F X6 = X1 + X2
F X6 = X4 * X5

Both instructions require X6 for their result. In this case, the Floating Add result would be returned to X6 before the Multiply result was desired.

There are then, two types of First Order Conflicts - functional unit and result register. In all cases of first order conflicts, issuance of instructions stops until the conflict is resolved. In other words, no further instructions are initiated (including the one which "sees" the conflict) until the first of the conflicting instructions has completed. In conclusion, first order conflicts temporarily stop issuance of instructions at the point of conflict.

2) SECOND ORDER: A conflict that occurs when an instruction requires the result register of a previously initiated instruction as a source operand.

EXAMPLE:

F (X6) = X1 + X2
F X7 = X5 / (X6)

In this case, the Divide unit needs X6, which is the result of the Add instruction, as one of its source operands. The Divide Unit must obviously wait for the Add unit to time out, but instruction issue will not stop. Instead, the Scoreboard will delay the start of the Divide instruction until the Add unit has stored its result. Subsequent instructions may be issued as long as no First Order conflicts exist.

The result of a Second Order Conflict is to delay the execution of the conflicting instruction only.

3) THIRD ORDER: A conflict that occurs when one instruction must store its result in a register which is to be used as a source operand for a previously issued instruction.

EXAMPLE:

F X3 = X1 / X2
F X5 = (X4) * X3
F (X4) = X0 + X6

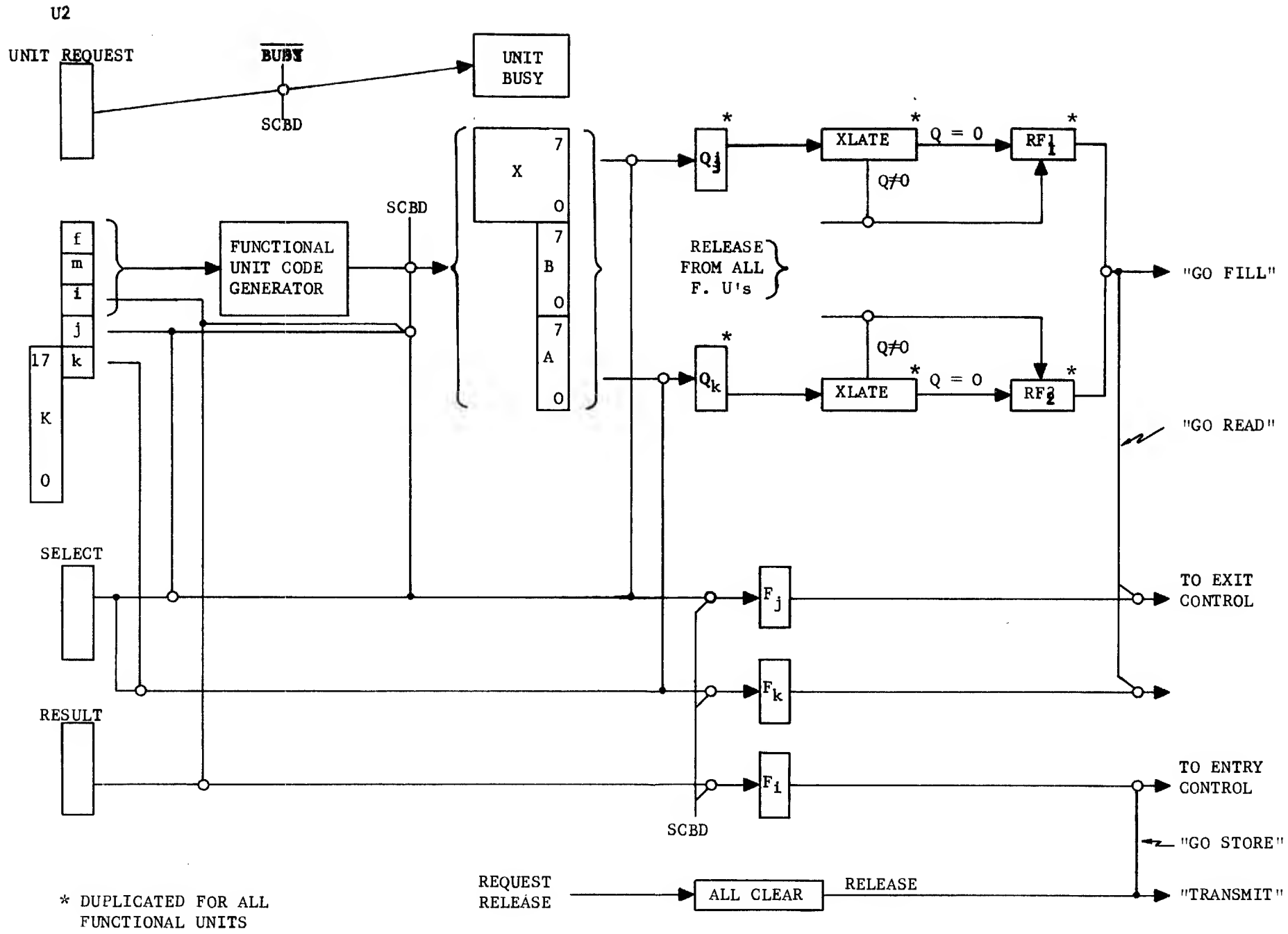
In this example, due to the relatively long execution times of the Divide and Multiply op. codes and the second order conflict (X3) of these units, the Add instruction will complete its calculation before the Multiply unit has read its operands (both operands are always read at the same time; therefore, all second order conflicts must be resolved). Since the Multiply instruction is intended to read X4 before it is changed by the Add instruction, storage of the Add result must be delayed until the Multiply Unit begins its calculation. Thus, third order conflicts do not delay issue or calculation, but rather the storage of a result operand.

The following discussion explains, at the block diagram level, how these conflicts are handled by the scoreboard. Figure 1-8 should be used in following the explanation.

FIRST-ORDER CONFLICTS

First-order conflicts are defined as either functional unit or result register conflicts. If either type exists, no issues can be generated.

Functional unit conflicts are determined by checking the Unit Request flip-flops against the Unit Busy flip-flops. Recall that Unit Requests are set at the



SCOREBOARD BLOCK DIAGRAM
Figure 1-8

time an instruction is transferred from U1 to U2. The Unit Request flip-flop sets depending upon the op.code that is translated (from U1 fmi portion). When an instruction is issued to the scoreboard, among other things, a Unit Busy flip-flop will be set. This in effect, reserves a functional unit for a particular op.code - it will remain reserved until the given instruction has completed execution, at which time the Unit Busy flip-flop is cleared to allow a subsequent instruction to use that functional unit. Thus, when a given unit is needed (as determined by a set Unit Request flip-flop) it can be used only if the associated Unit Busy flip-flop is cleared. If the Unit Busy is set, a functional unit conflict exists and generation of issues is disabled until the unit is freed.

In determining the existance of result register conflicts, a comparison of the request and reservation logic is also made. Upon issuing an instruction to U2, Result flip-flops are set according to the op.code translation. Four such flip-flops exist and specify a result register group (i.e. Xi, Bi, Ai or Bj). The specific register within a group is determined by translating the i or j octals (as specified by the Result flip-flops). For example, if the Xi flip-flop is set and the U2 i digit = 3, X3 is the result register desired.

The result register reservations are placed in the "XBA reservation list" when an instruction is issued to the scoreboard. This list is composed of 24 "slots", where-in codes are placed to specify which functional unit has reserved each of the 24 operating registers. For example, a code of $16_{(8)}$ in the X4 slot of the reservation list indicates that X4 is reserved for the result of the LONG ADD functional Unit. The complete list of possible codes follows:

UNIT	CODE
Increment 1	01
Increment 2	02
Shift	03
Boolean	04
Divide	05
Multiply 1	06
Multiply 2	07
Read Memory, Channel 1	11
Read Memory, Channel 2	12
Read Memory, Channel 3	13
Read Memory, Channel 4	14
Read Memory, Channel 5	15
Long Add	16
Add	17

Any slot that contains an all-zero code indicates that the associated operating register is not reserved. Any non-zero code indicates that the associated register is reserved for a result. Thus, if translation of U2 indicates that X4 is specified as a result register and the X4 slot of the reservation list is zero , no conflict occurs. If the X4 slot is not equal to zero, the register is reserved. Thus, a conflict exists and issues are disabled until the conflict is resolved.

Notice from the list of codes that 5 are named Read Memory, Channel X . These are necessary for the 5 X 1 - 5 X 5 instructions. They return results from Memory to X 1 - X 5 and must make a result register reservation. In this sense, Memory acts like a functional unit.

In summary, both cases of first-order conflicts are handled similarly in that requests for units or result regsiters (made at U2 time) are checked against reservations existing in the Scoreboard. If a conflict exists, issues are disabled until the conflict is resolved.

SECOND-ORDER CONFLICTS

Second-order conflicts occur when a functional unit requires as a source operand, the result of another functional unit. The source operands are defined by the j and k octals of U2 in conjunction with the select flip-flops which are set with a U2 transfer. The select flip-flops define the source register group as well as the octal digit specifying the register within that group (i.e. Xj, Bj, Aj, Bk or Xk). By ANDing select flip-flops with the j and k octal digit translations, specific registers are selected.

Determination of whether or not the desired registers are reserved is made by looking at the content of the XBA reservation list, but not directly. Each functional unit has 4-bit Q designators which, when an instruction is issued to the scoreboard, receive the contents of the XBA slot associated with the desired source operand registers. For example, the following instruction sequence causes a second order conflict:

```
F X 5 = X3 * X2
F X 6 = X2 + X5
```

The Multiply I unit reserves X5 by placing a code of $06_{(8)}$ in the X5 slot of the reservation list. Assuming that no other instructions have been issued, no other reservations exist when the Add instruction is issued. Since the Add unit wished to read X2 and X5, it transfers to its Qj and Qk designators the content of the X2 and X5 slots, respectively. At this point, the Add Qj designator equals $00_{(8)}$ and Qk equals $06_{(8)}$. In essence, this tells the Add unit that its j operand (X2) is not reserved by the Multiply I unit. Since a functional unit does not begin calculation until after it can read both operands, the Add unit must wait until Multiply I returns its result to X5.

Associated with each functional unit are flip-flops, called Read Flags , which when set, indicate that the desired operand (s) can be read. The Add unit has two Read Flags, one for the Xj operand and one for Xk. Read Flags can be set in two ways, both of which result from translating the Q designators.

1) If $Q = 00_{(8)}$, a Read Flag can be set since the desired operand is not reserved.

2) If $Q \neq 00_{(8)}$, a Read Flag cannot be set until the functional unit, whose code is in Q, has completed its calculation and returned its result to the result register. Completion of a functional unit's operation is indicated by a signal called Release (discussed in detail under third order conflicts).

In the above example, the Add unit's Xj Read Flag is set immediately, since Qj = 0. The Xk Read Flag is set when the Release for Multiply 1 occurs, since Qk translates as $06_{(8)}$. Each possible non-zero Q translation is tied to the "Release" signal for the associated functional unit, so it is possible to set a Read Flag by any translation of Q, ANDed with the associated "Release" signal or, by $Q = 00_{(8)}$.

Once both Read Flags are set, it is necessary to send the functional unit its operands and to send a Go signal to the unit, allowing it to begin its calculation. The Go F.U. signal is sent as soon as both Read Flags are set. This signal starts the functional unit timing chain. At the same time, the source register selection codes are sent (by a Go Read signal) to Register Exit Control to gate the proper operands to the unit. These codes are obtained from the F designators associated with each unit. These are 3-bit designators which are used to remember the source and result operand register numbers. They also are set when the scoreboard is issuing an instruction. In the above example, once both Read Flags are set, the content of the Fj and Fk designators of the Add unit are sent to Register Exit Control and will allow X2 and X5 to be gated to the Add unit. The Read Flags are cleared during the minor cycle after both are set. Set Read Flags then, indicate that an operand is waiting to be read.

Thus, the general second-order conflict case delays the start of a functional unit until both source operands can be read. Some special cases exist, which are discussed in detail in the logic analysis sections of this manual. At this point, it is appropriate to understand the general case.

THIRD-ORDER CONFLICTS

The possibility of third-order conflicts occurs when a functional unit has generated a result and wishes to store in an operating register. If the desired result register is waiting to be read, the unit must wait to store until after the read has occurred.

Whether or not a register is waiting to read is determined by checking the Fj and Fk designators against the associated Read Flags in all the functional units. The result register of a unit is given by the Fi (and, in some cases, Fj) designator of that unit. When a unit requests to store a result, its result register number is checked against the Read Flags and F designators of all other units. If any Read Flag is set AND the associated Fj or Fk designator translation is the same as the Fi designator of the storing unit, a third-order conflict exists. The unit will therefore be prevented from storing until the conflicting unit's Read Flag is cleared. This, of course, occurs once a unit has set both of its Read Flags.

The general sequence in handling third-order conflicts is as follows. First, a unit desiring to store a result sends a Request Release signal to the scoreboard near the end of its calculate time. This signal is then ANDed with an All Clear signal to generate the Release gate, which allows storage of the result. The All Clear is the result of checking all Read Flags with the associated Fj and Fk designators and comparing with the Fi (or Fj) designator (for the result) of the unit requesting release. The Release signal accomplishes several necessary tasks in the scoreboard. It sends a transmit signal to the functional unit to gate the result to the data trunk. It also generates a Go store signal which gates the Fi (or Fj) designator to Register Entry Control to select the desired result register. Release also clears reservations in the scoreboard (i.e. XBA designators, Unit busy flip-flops, etc.) and checks all Q designator translations in the event that a unit is waiting to use this result as a source operand. The Release then, indicates final termination of an instruction, and in essence, removes that instruction from the scoreboard.

REGISTER EXIT/ENTRY CONTROL

As the name implies, Register Exit/Entry Control is the control logic used for gating data into and out of the 24 operating registers. It is in essence, a large translating network which decodes tags sent from the Stunt Box or Scoreboard to enable the transfer of data to and from central memory or the functional units. Figure 1-9 is a block diagram which should be used during the following discussion.

ENTRY CONTROL

Entry Control is shown on the left half of Figure 1-9. To the extreme left are the four general sources of information for the X, B and A registers:

1) Central Memory

2) Data Trunk #1 (Shift, Add and L. Add)

3) Data Trunk #2 (Boolean, Divide, Multiply 1 and Multiply 2)

4) Data Trunk #3 (Increment 1 and 2)

Data is entered into the operating registers from Central Memory during Exchange Jumps and during the central read operand instructions (5 X 1 - 5 X 5). Since memory references are involved, all of the gating tags are sent from the Stunt Box and are composed of the lower four bits (2^0 - 2^3) of the Hopper Tag ANDed with the Accept signal for the associated address. In other words, when an Exchange Jump or a Read Operand address is accepted, the four-bit tag is sent from the tag timing chain to Entry Control where it enables the information from Central Memory to the proper X, B, or A register.

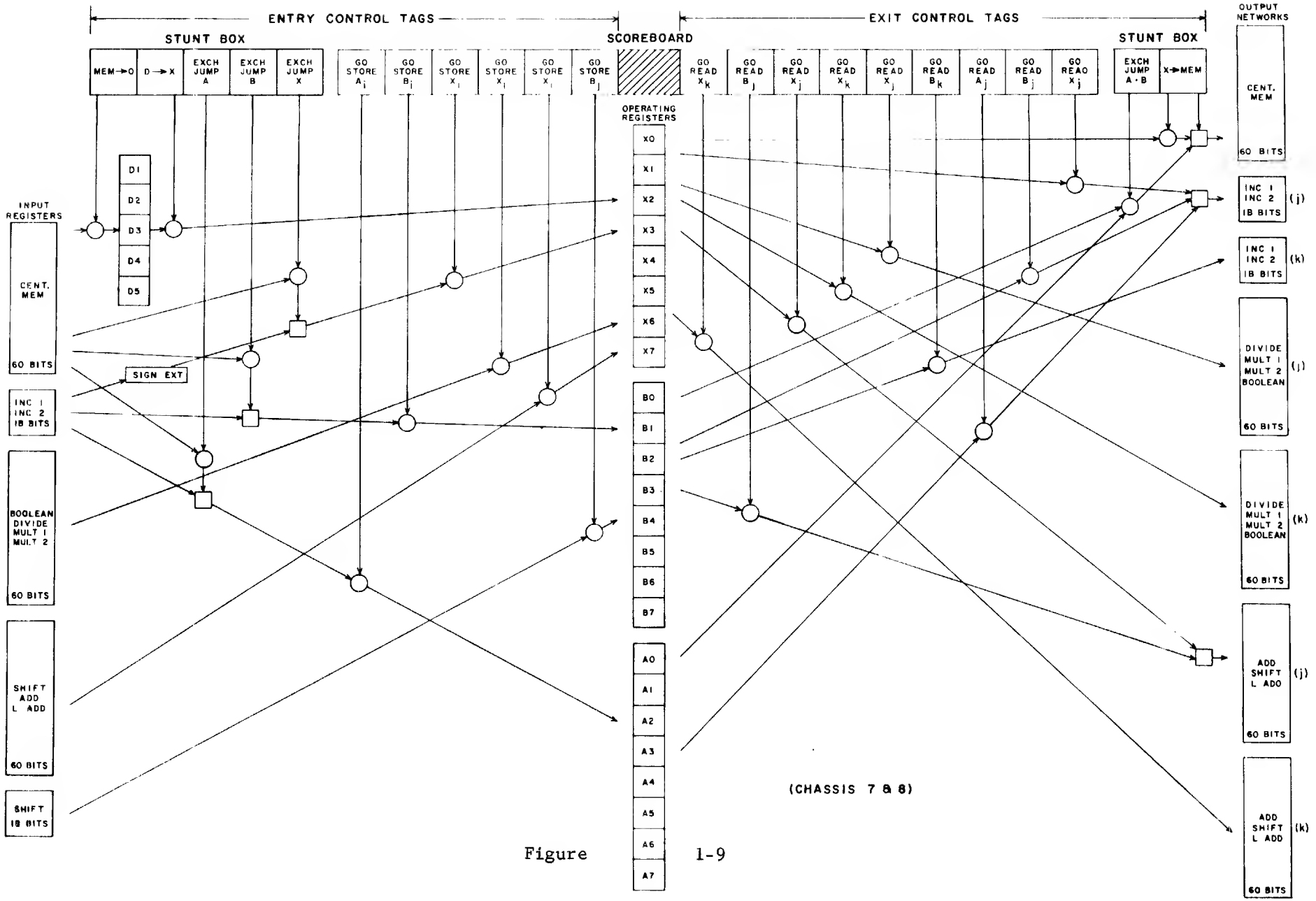

ENTRY CONTROL TAGS
EXIT CONTROL TAGS
STUNT BOX
SCOREBOARD
STUNT BOX
OUTPUT NETWORKS
MEM→0
D→X
EXCH JUMP A
EXCH JUMP B
EXCH JUMP X
GO STORE Ai
GO STORE Bi
GO STORE Xi
GO STORE Xi
GO STORE Xi
GO STORE Bj
GO READ Xk
GO READ Bj
GO READ Xj
GO READ Xk
GO READ Xj
GO READ Bk
GO READ Aj
GO READ Bj
GO READ Xj
EXCH JUMP A·B
X→MEM
CENT. MEM 60 BITS
INC 1 INC 2 18 BITS (j)
INC 1 INC 2 18 BITS (k)
DIVIDE MULT 1 MULT 2 BOOLEAN 60 BITS (j)
DIVIDE MULT 1 MULT 2 BOOLEAN 60 BITS (k)
ADD SHIFT L ADD 60 BITS (j)
ADD SHIFT L ADD 60 BITS (k)
INPUT REGISTERS
CENT. MEM 60 BITS
INC 1 INC 2 18 BITS
BOOLEAN DIVIDE MULT 1 MULT 2 60 BITS
SHIFT ADD L ADD 60 BITS
SHIFT 18 BITS
D1
D2
D3
D4
D5
SIGN EXT
OPERATING REGISTERS
X0
X1
X2
X3
X4
X5
X6
X7
B0
B1
B2
B3
B4
B5
B6
B7
A0
A1
A2
A3
A4
A5
A6
A7
(CHASSIS 7 & 8)

Figure 1-9

Notice that during Read Operand references, the 60-bit operand is first sent into the D register (1 - 5) associated with the X register (1 - 5) which will ultimately receive the information. This is enabled by the Mem $\longrightarrow$ D signal which results from the simple translation: (tag 11 - 15) (Accept). The operand will be temporarily stored in D, until any third order conflict which may exist is resolved. (A Read Flag may be set for the X register which is to receive the operand from memory). Thus, when the All Clear signal occurs, the D $\longrightarrow$ X signal is generated and completes the transfer to X.

During Exchange Jumps, new information is entered into A, B and X registers by Hopper tags in the range, 60 - 77. Recall, that tags 60 - 67 enable the exchange of A and B registers, while tags 70 - 77 enable exchanging X registers. The Entry Control Tags, Exchange Jump A, Exchange Jump B and Exchange Jump X refer to the hopper tage 60 - 77 accepted.

A result generated by the Increment units may be entered into X, B or A registers, depending upon the instruction being processed (5X, 6X, or 7X). Thus, three Entry Control Tags are shown for the Increment Data trunk, namely, Go Store Ai, Go Store Bi and Go Store Xi. Recall that Go Store occurs after a functional unit has been released and enables the Fi designator content to Entry Control. Thus, the Go Store tags are generated by the Scoreboard at the completion of an instruction sequence. Note also, that Sign Extension occurs when storing an Increment result (18 bits) in an X register (60 bits).

Results generated by the Boolean, Divide, Multiply 1 or Multiply 2 units are always 60 bits in length and the result register of these functional units is always an X register. Therefore, one Entry Control Tag, namely Go Store Xi is shown for Data Trunk #2. The tag is also generated from the Fi designators of the units on this trunk when the unit is Released by the Scoreboard.

The units on Data Trunk #1, Shift, Add, and Long Add all generate a 60-bit result for X registers, but in addition, the Shift unit may generate an 18-bit result for a Bj register. (For example, during normalize or unpack operations.) Thus two Entry Control Tags are shown for this trunk: Go Store Xi and Go Store Bj. These are also generated by the Scoreboard when a unit releases from the Fi (or Fj in the special shift case) designators.

EXIT CONTROL

Similar to Entry Control, Exit Control has four general destinations for data from the operating registers:

1) Central Memory

2) Data Trunk #1

3) Data Trunk #2

4) Data Trunk #3

The Exit Control Tags are also generated similarly, that is, from the Stunt Box or the Scoreboard.

The Stunt Box generates tags for information to be sent to central memory, specifically, during Exchange Jumps or Central Store Operand instructions (5 X 6 or 5 X 7). During these operations, the lower four bits of the Hopper Tag are sent to Exit Control when the associated address has been Accepted (is not in conflict). The data, which may be A and B register or X register contents, are sent on the memory trunk and will be stored during the write portion of the memory cycle.

For the Increment Data Trunk, four Exit Control Tags are shown in Figure 1-9 since the Increment units may specify an A, B or X register with the j octal and only a B register with the k octal. Thus the four tags, Go Read Xj , Go Read Bj , Go Read Aj , and Go Read Bk are used to gate operands on this trunk.

For Data Trunk #2, all functional units specify only X registers as source operands. Therefore, only Go Read Xj and "Go Read Xk" tags are required.

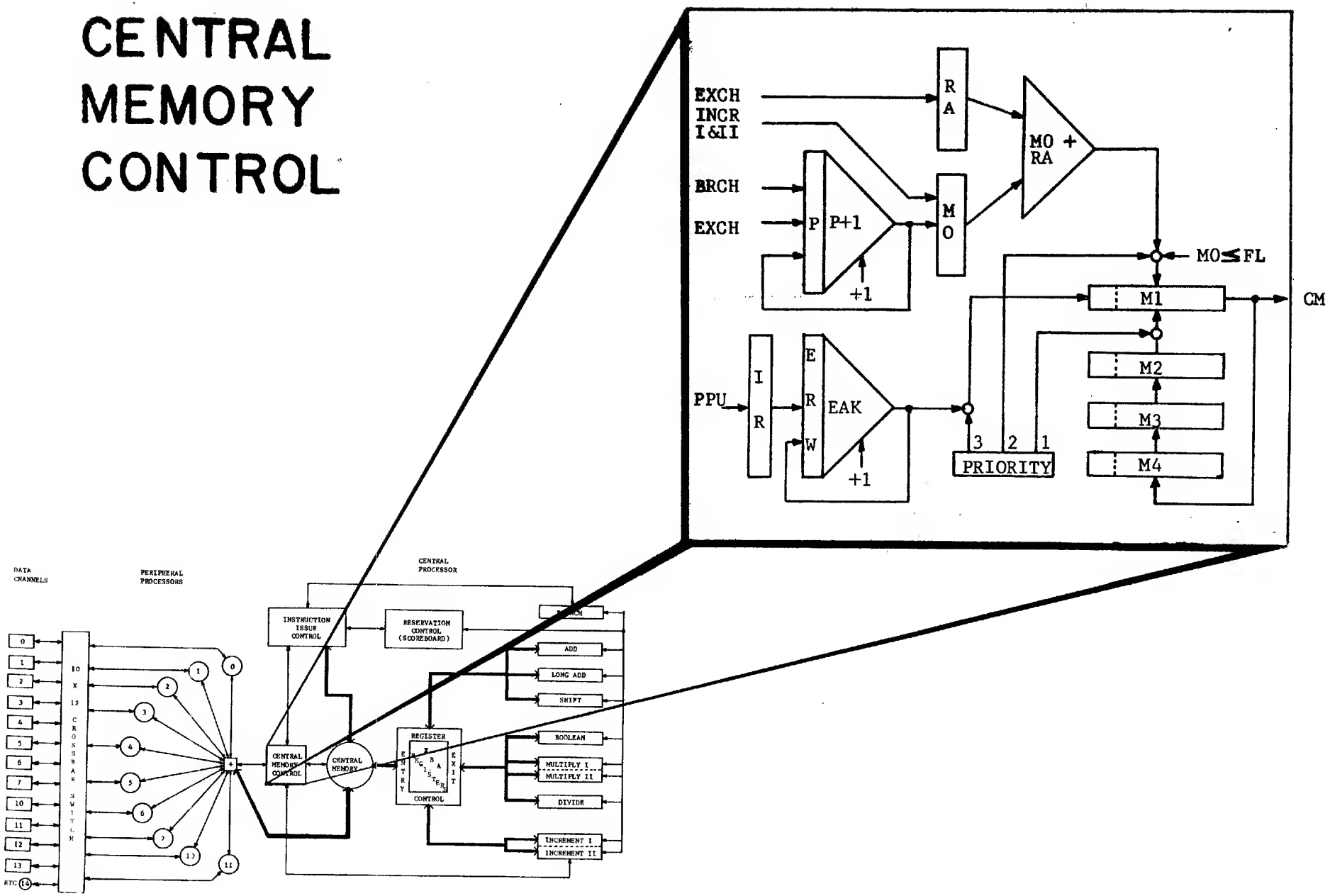
For Trunk #1, the Add, and L. Add units may specify an Xj or an Xk source register (or both) while the shift unit may specify an Xj or Bk register (or both). Thus the three tags, Go Read Xj , Go Read Bj and Go Read Xk are required for this data trunk.

All Go Read tags are generated by the Scoreboard when both Read Flags for a unit have been set. This enables the Fj or Fk designator to exit control and gates the proper register to the proper trunk.

CHAPTER II

CENTRAL MEMORY ADDRESS CONTROL

CENTRAL MEMORY CONTROL



CHAPTER II

CENTRAL MEMORY ADDRESS CONTROL

INTRODUCTION

The basic concept of the 6600 computer is parallelism. In the central processor many parallel operations can be in progress, one of which may be accessing of central memory. Sometimes the peripheral processors will be attempting communications with central memory simultaneously with the central processor. This is a simplification of possibilities but points out that some method of orderly distribution of requests for central memory time must be available.

Central Memory is divided into 32 (or 16) independent banks that can be accessed sequentially in an overlapping fashion. This allows memory references every 100 nano-seconds. To make full use of central memory and also to fulfill the parallelism concept, an area of the central processor known as the STUNT BOX exists. It is to this circuitry that all references to central memory must first come. The job of the stunt box is to collect requests for central memory access and to distribute these requests in an orderly fashion.

The 6600 has ten peripheral processors that can request read or write operations in central memory. The peripheral processors can also execute an exchange jump which requires central memory access. The central processor can reference memory for an instruction word or to read and store operands.

Consequently the peripheral and central processors can send requests to the stunt box simultaneously. This presents the necessity for some means of priority within the stunt box and also some method of naming each of the memory requests in order that the data, once acquired, will be distributed correctly. Another factor that must be considered is a means of remembering addresses if the first attempted access is rejected because of a memory conflict.

In general, the stunt box:

a) allows several simultaneous memory requests.
b) establishes a priority for issuing addresses to central memory.
c) issues the addresses to memory at a rate that will make maximum use of the 32 (or 16) independent banks.
d) remembers addresses that have not been accepted by the memory and must be re-issued.
e) adds a tag to the addresses to correctly distribute the data.

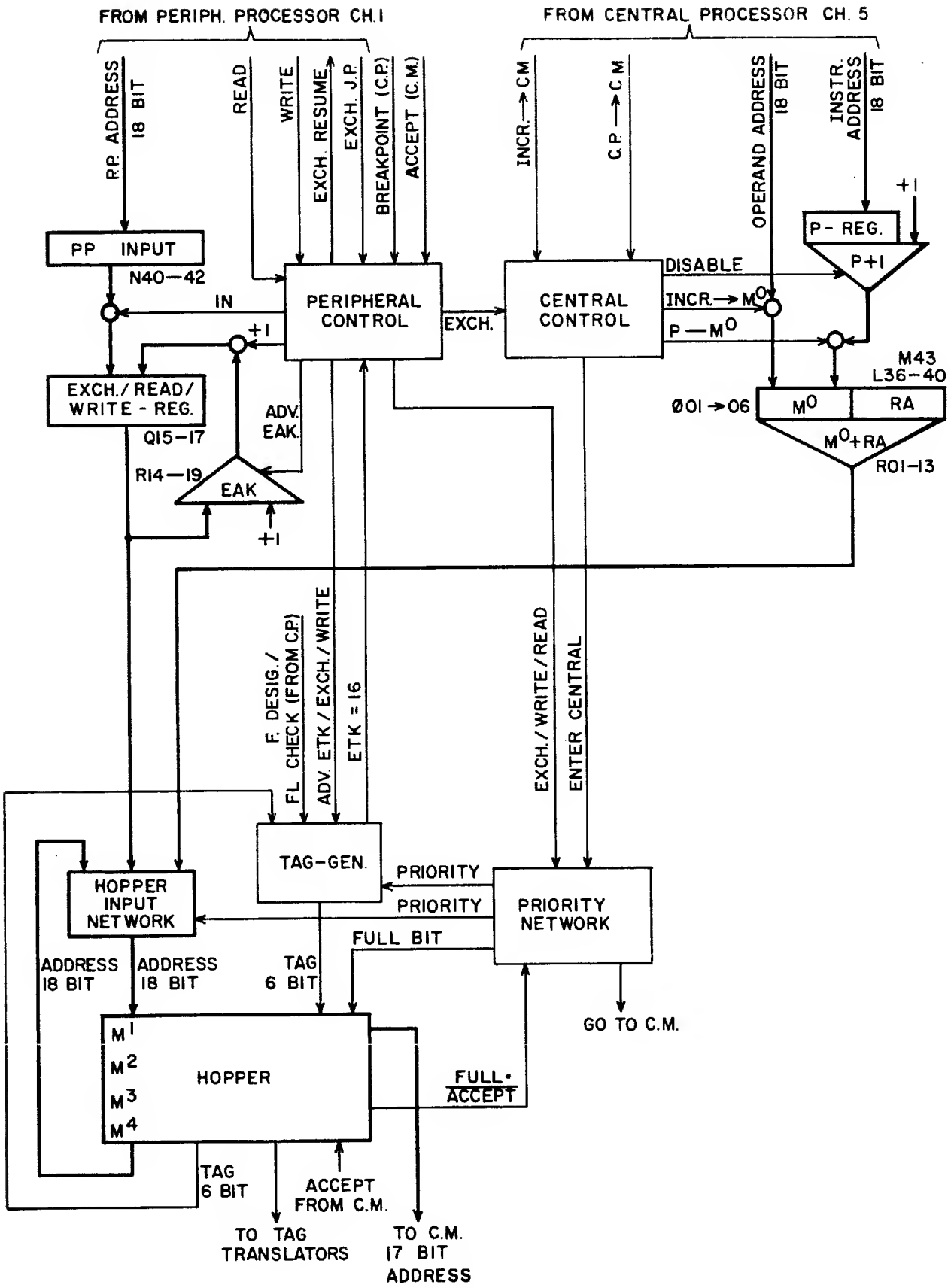


Figure 2-1. Stunt Box Block Diagram

STUNT BOX LOGIC ANALYSIS

DESCRIPTION

There are two main address flow paths to the hopper (M_1) and to Central Memory (CM): (Refer to Figure 2-1).

1) The PPs supply CM with the write and read addresses for the data-exchange between PPs and CP. The exchange jump addresses are also sent over this line.

2) The central processor supplies CM with either operand or instruction addresses. Addresses comming through this path are always added to the content of the Reference Address (RA) Register.

A priority network controls the entry into the hopper (M_1) from which the addresses are sent to CM. These addresses are also stored and circulated in the hopper, from which (in case of bank conflict) they are re-issued to CM after 300 nanoseconds.

Peripheral Control

Peripheral Control, for the purposes of this discussion, is a term applied to that section of the central processor that handles addresses and signals from the peripheral processors requesting storage access. (Refer to the block diagram in Figure 2-2)

When the peripheral processors send an address to the stunt box, an accompanying signal informs Peripheral Control whether it is a Read, a Write or an Exchange Jump address. Peripheral Control (PC) then transfers this information to the tag generator to enable the IN-path to the EXCH/READ/WRITE Register. In an Exchange Jump, this signal also stops the central processor: When the :Breakin" signal indicates the central processor and central memory have stopped, it starts Exchange Address (EAK) and Exchange Tag Counters (ETK). The Exchange Address Counter then updates the address in the EXCH/READ/WRITE register for each step of the Exchange process upon receipt of the "accept" from central memory. When the Exchange Tag Counter = 16, Peripheral Control sends an Exchange Resume back to the peripheral processors (Write Resume is sent back from central memory chassis 2, and Read Resume from chassis 4).

Central Control

Central Control, for the purposes of this discussion, is a term applied to that section of the central processor that handles addresses from the central processor requesting access to central memory for instructions or operands. Central Control controls entry into M0 and requests entry (via the priority network) into M1. Refer to

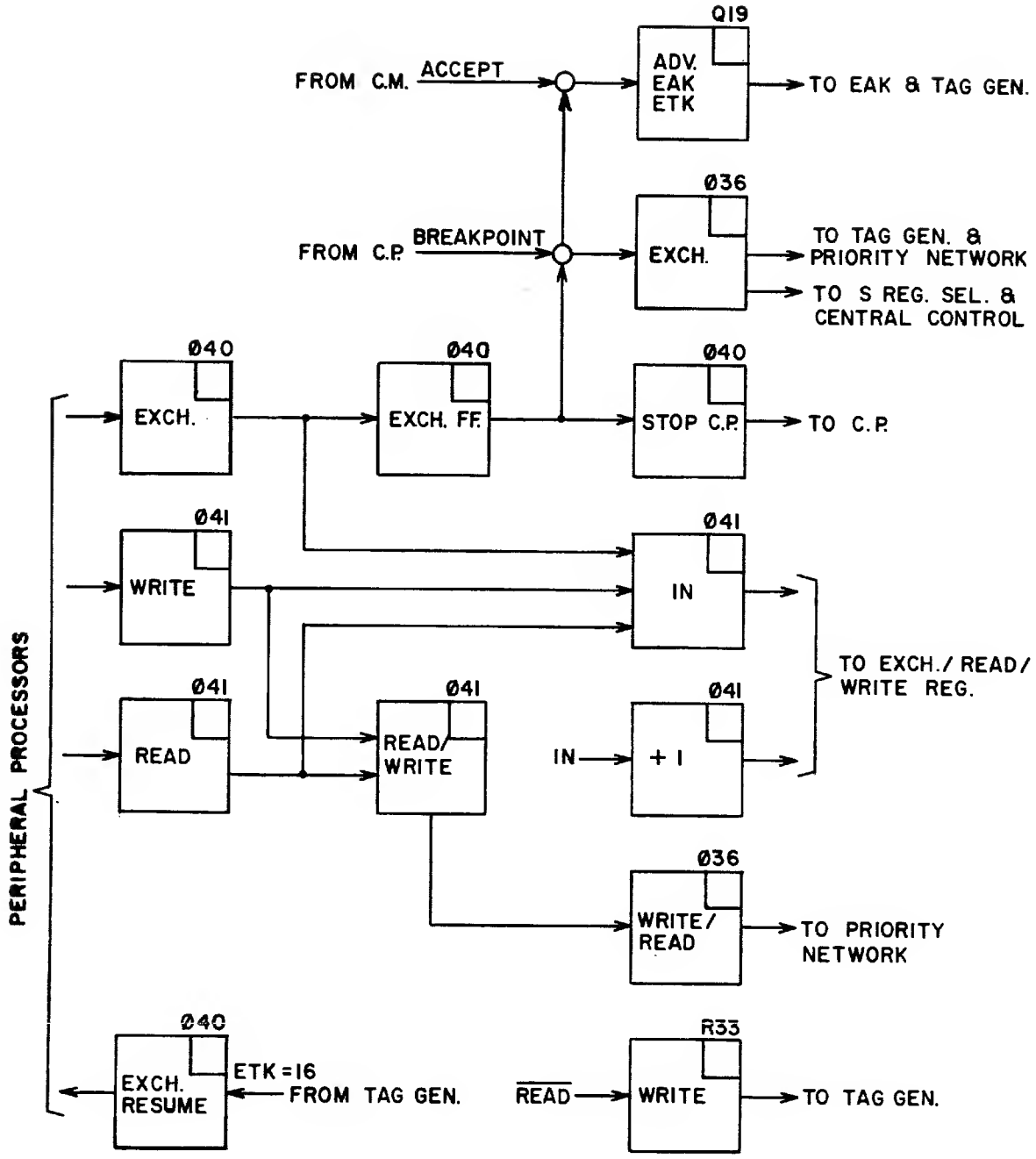


Figure 2-2. Peripheral Control Block Diagram

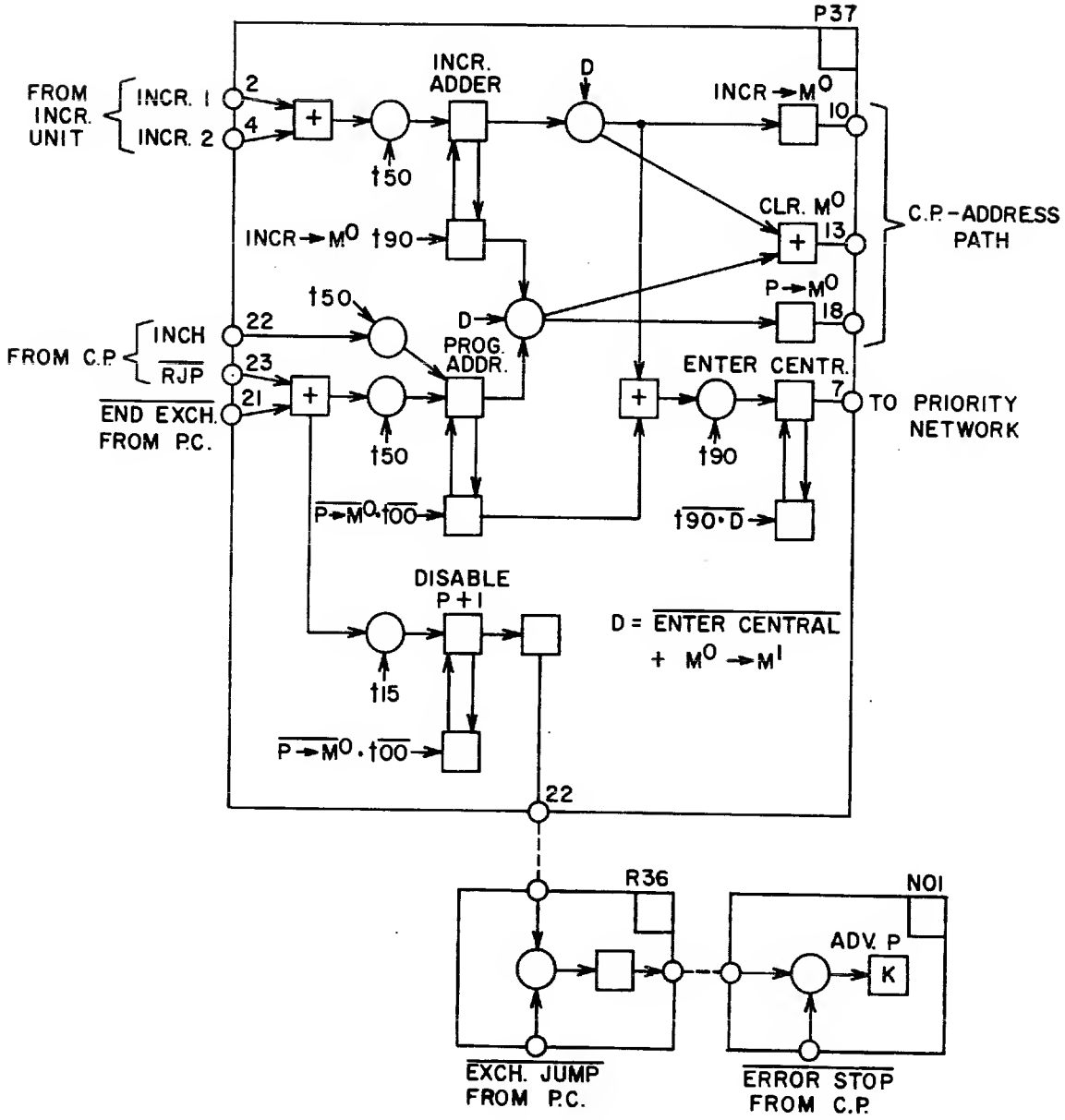


Figure 2-3. Simplified Central Control Network

Figure 2-3)

When an operand address is sent by one of the two increment units, the INCREMENT ADDRESS FF on P37 will set and the entry signal into M_0 is enabled as soon as M_0 is empty. Simultaneously, the ENTER CENTRAL FF sets indicating to the priority network that a central processor address is waiting to enter central memory.

When a program address is ready in P, the PROGRAM ADDRESS FF sets, enabling the $P \longrightarrow M_0$ signal if M_0 is empty and if no operand address is waiting. This signal also sets the ENTER CENTRAL FF.

If a Return Jump instruction or an End Exchange signal occurs, the DISABLE P + 1 FF sets and the program address in P passes through the P-incrementor without being incremented. This does not affect the ADVANCE P gates on N37-N39.

PRIORITY NETWORK

The priority network (Figure 2-4) controls inputs to the hopper by sequencing entry to the hopper when more than one address attempts to enter at the same time.

The fixed order of priority is as follows:

1) Address from Hopper
2) Addresses from the Central Processor
3) Addresses from the Peripheral Processors

An address from the hopper is given first priority since it is an un-accepted address resulting from a central memory bank conflict.

Addresses with second priority are from the central processor (i.e., M^0). Since, for the central processor, storage modes cannot be mixed in the hopper, the Read or Write tags are examined before priority is granted. In attempting a Read, no Write address is allowed in M^1 or M^4. In attempting a Write, no Read address is allowed in M^1 or M^4. If modes are mixed, priority is not granted and entry of the address into the hopper is delayed until central memory has accepted those addresses and modes are no longer mixed.

Addresses from the peripheral processors are assigned lowest priority. Thus, peripheral read and write operations from and to central memory may have to wait for hopper and central processor addresses. An important exception occurs during an Exchange Jump. An Exchange Jump a) stops the central processor, and b) inhibits communications between the peripheral processors and central memory. In this case, exchange jump addresses are the only addresses entering the hopper.

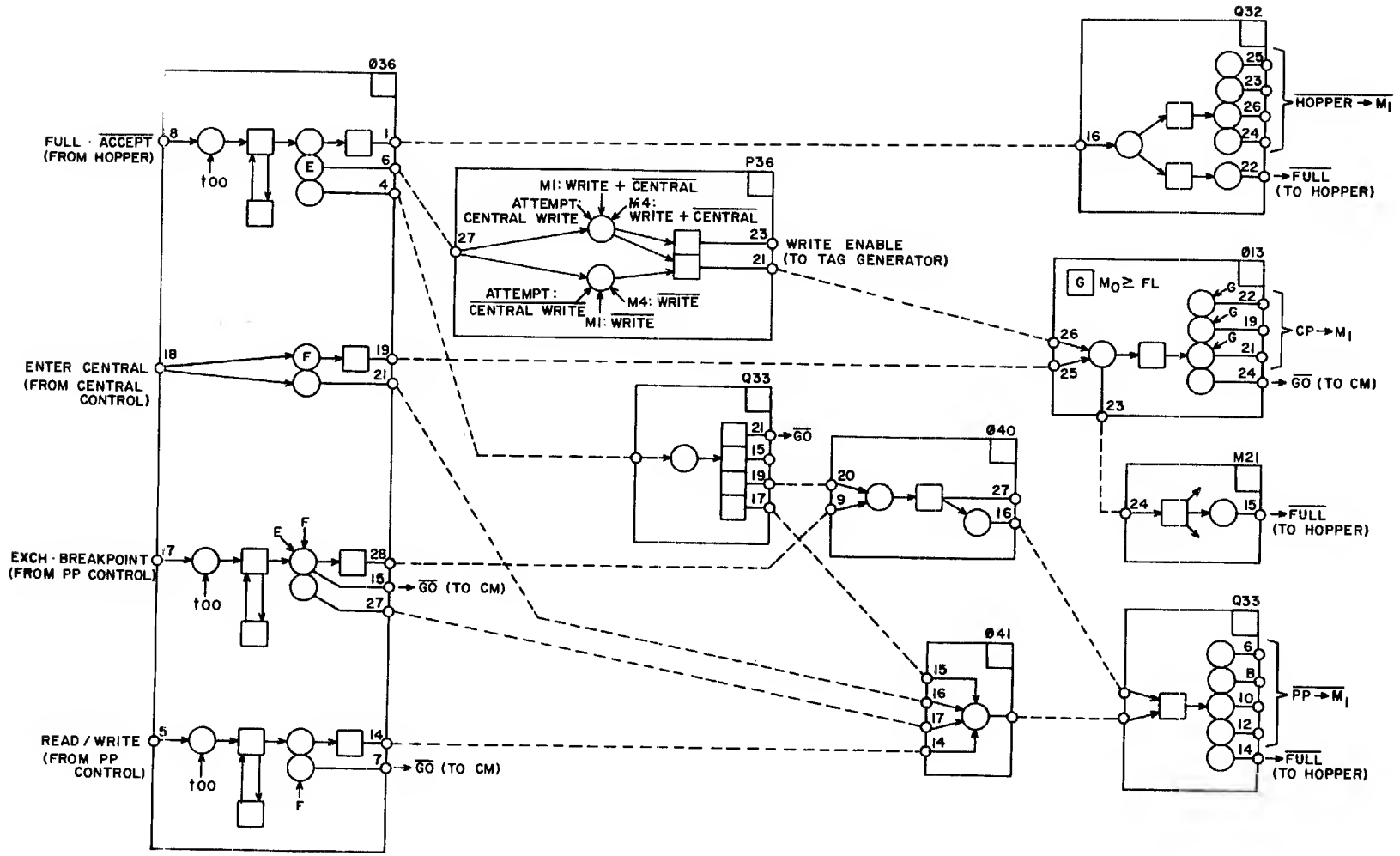


Figure 2-4. Priority Network

With each address sent to the hopper, a Full bit is generated, (to indicate the hopper register contains a usable address a tag and a Go signal is sent, along with the address, from M^1 to central memory.

The hopper input network is diagrammed in Figure 2-5. Three gates for each of the 18 address bits accommodate the three possible input paths to the hopper. Entry via these gates is controlled by the priority network.

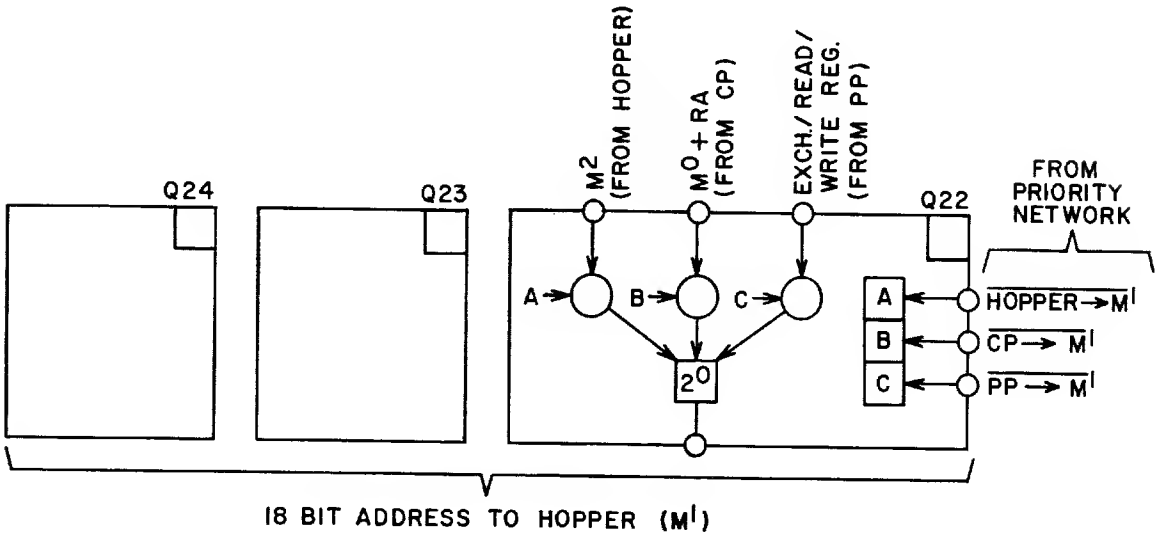


Figure 2-5. Hopper Input Network

TAG GENERATOR

When an address enters the hopper, a six-bit tag is appended to control the address and data flow. Depending on the source of the addresses, these tags are generated from three sources: (Refer to Figure 2-6)

1) An un-accepted address resulting from a bank conflict retains the tag that was generated when the address first entered the hopper.

2) An operand or instruction address from the central processor gets its tag from the translation of the F designators of the increment units or from the central processor (in case of exit mode stops or return jumps) and from the priority network.

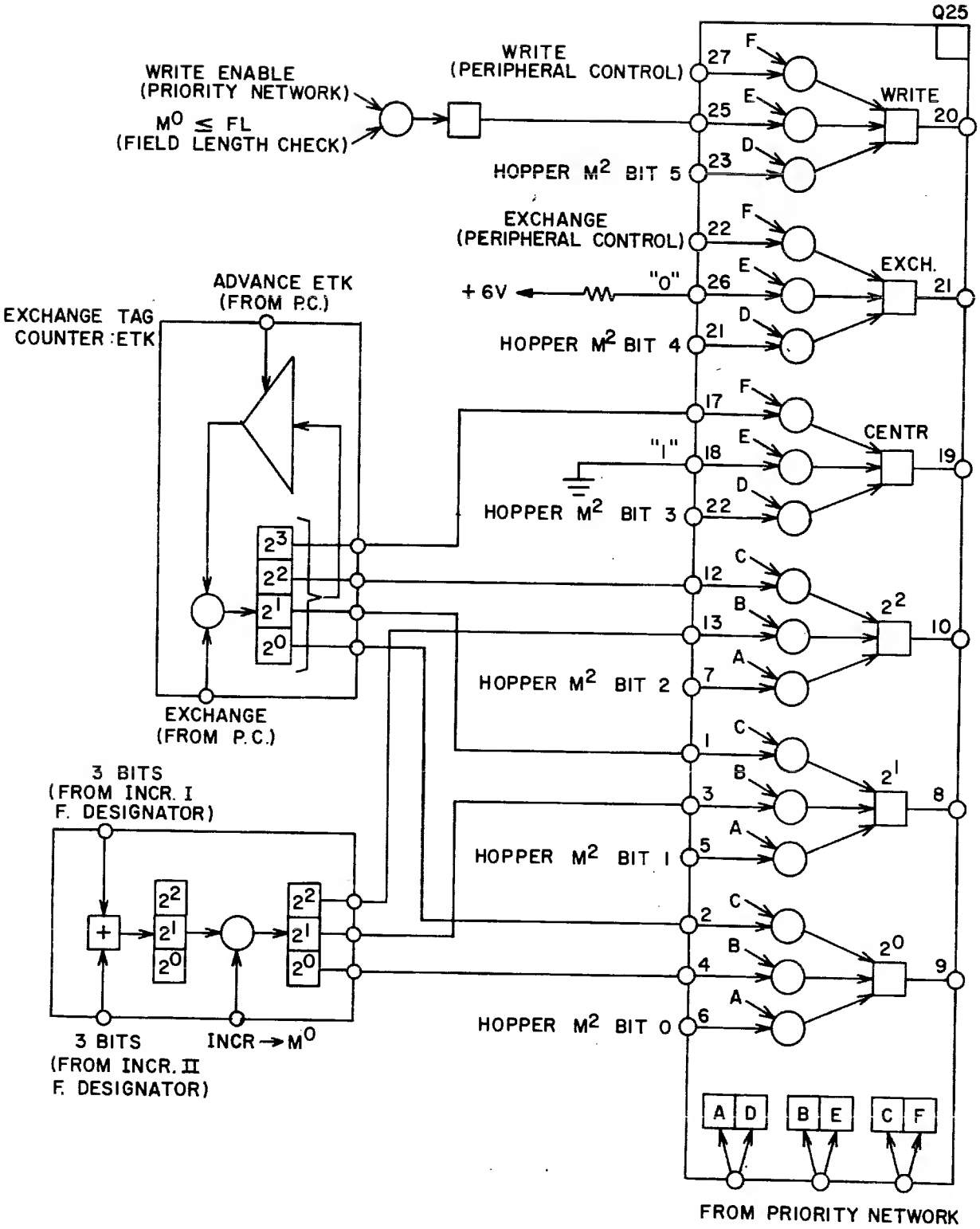


Figure 2-6. Tag Generator

3) Addresses from the peripheral processors obtain their tags from Peripheral Control. The Exchange Tag Counter (ETK), which controls the execution of the exchange jump, generates the tags for all addresses of the exchange jump package.

HOPPER

The hopper consists of four registers (M^1, M^2, M^3, and M^4) each capable of holding an 18-bit address, a 6-bit tag, and a single Full bit. (M^2 is an exception and does not have a Full bit). (A block diagram of the hopper is shown in Figure 2-8)

An address is sent to central memory from hopper register M^1. Hopper registers M^2 - M^4 store the address in case it must be re-issued because of a bank conflict. If the address is accepted by central memory, it drops out of M^2.

In the case of bank conflict, the priority network gates the un-accepted address from M^2 back into M^1 every 300 nanoseconds, until it is accepted by central memory. An address can be accepted only if the specified bank is free at the time the address is in M^1. Otherwise, it is possible for another address to request access to the same bank and tie it up for a memory cycle. Figure 2-7 shows possible waiting times in the hopper for worst cases of bank conflict.

The six tag bits travel through the hopper with each address. (The hopper serves as a delay line for the tag.) This line is extended by two additional registers (see Figure 2-8) In each step, the tag controls address and data flow.

Hopper registers M^1, M^4, and M^3 have Full bits associated with the address. The purpose of the Full bit is to indicate to the priority network that the address must be reissued to central memory if no accept is returned. Note that the Full bits also are sampled to stop the central processor (Figure 2-8) before intiating an exchange jump (Break IN).

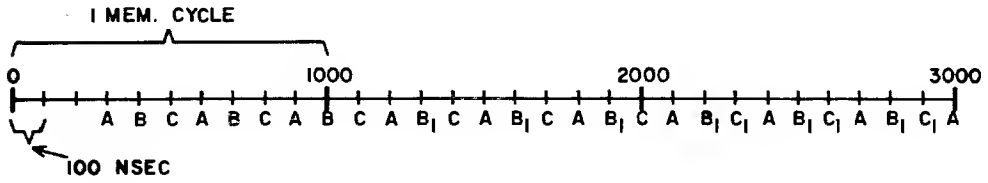


Figure 2-7. Hopper Waiting Time for Bank Conflicts

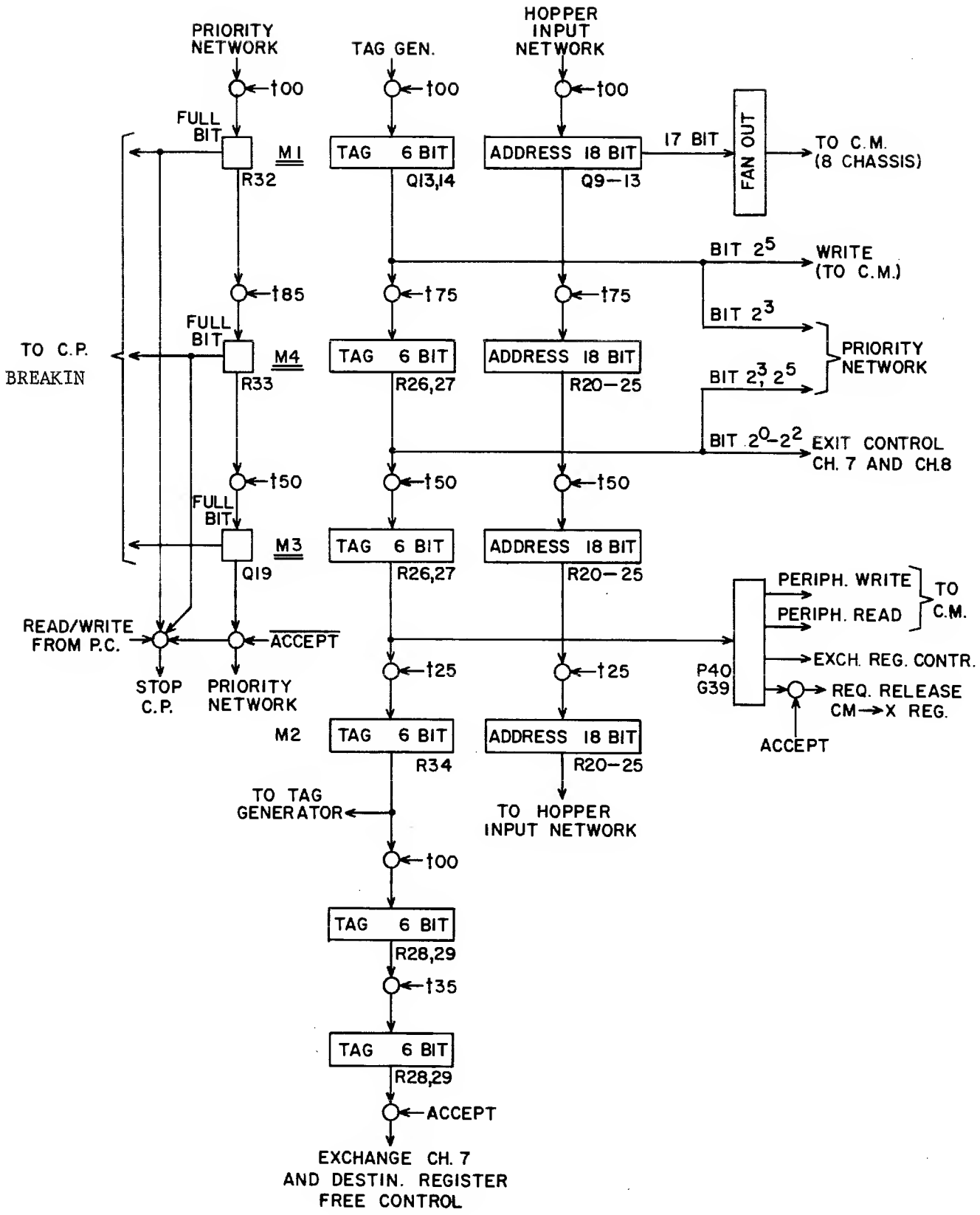


Figure 2-8. Hopper

The Hopper can contain a maximum of 3 addresses. An address must be in M1 at the time the memory cycle for its bank finished so that it can be accepted by memory, otherwise a bank conflict occurs. The longest time an address will have to wait to be accepted in memory is when the address enters M1 300 ns after the memory cycle started.

Example: Assume that all 3 addresses in the Hopper need the same bank, then at t=1000, address B is accepted and a new address B_1 can enter the Hopper, at t=2000 address C and at t=3000 address A will be accepted. (See timing diagram)

EXCHANGE JUMP

As an aid in following the somewhat detailed discussion which follows, refer to the central processor diagrams, the timing diagram in Figure 2-9 and the Exchange Jump diagram in Figure 2-10.

The following discussion assumes the Exchange Jump Package (see Reference Manual) is stored in Central Memory starting at address N. To start the Exchange Jump, a Peripheral & Control Processor will send an Exchange Jump pulse together with the 18 bit address N to the Stunt Box.

The address will be received by the Input Register (N40, 41, 42) and the Exchange pulse sets FF 040/TP5 at starting time 00. The "0" out of 040/P19 enables the A gates on Q27, 28, 29. At time 50 the contents of the Input Register are transferred to the Exchange/Read/Write Register Q15, 16, 17. At t65, 040/TP4 will be set, setting F37/TP3. If the computer is not running when the Exchange Jump is initiated, the AND gate next to F37/TP3 cannot be made because an Issue is not present, a "0" from F37/P24 entering I01/P21 and a "1" into I01/P26 are AND'ed giving a "0" into F37/P8 and therefore a "1" from F37/P7. If the computer is running the AND gate at the output of F37/TP3 is not made until an Issue signal occurs and the parcel counter = 1. The output of pin 10 then becomes a "0". This "0" clears the GO FF on G30, stopping the Central Processor. The "1" from F37/P7 will be passed on to F26/P12. If the Hopper is empty and no Branch or Register reservations are up a "1" occurs at F26/P10 making the AND gate and setting F26/TP4, placing a "0" into 040/P10. With the t75 pulse, 040/TP1 sets, disabling the A gates on Q27, 28 29.

This enables a "1" from 040/P13 to set the Initiate Exchange FF on 036/TP6 at t100. A "0" from 036/P15 and a t165 pulse sends a GO signal to central memory to prepare the path for the address being sent. Also at t200 a 60 tag from Q25 of the Exchange Tag Counter is sent to M^1.

The Initiate Exchange FF also sends a "1" to priority control, which, in turn, enables the C gates on Q22, 23, 24, 25. At t200, the contents of the Exchange/Read/Write Register are transferred to M^1 in the Hopper. During the time the address waits to enter M^1, a path from the Exchange/Read/Write Register through the Exchange Address Counter allows the address to circulate without being incremented until it can be accepted by M^1.

A bit called the Full bit is sent from the priority control to M^1. This bit is only used in the Hopper to indicate that the Hopper contains a usable address. The M^1 Register should now contain the

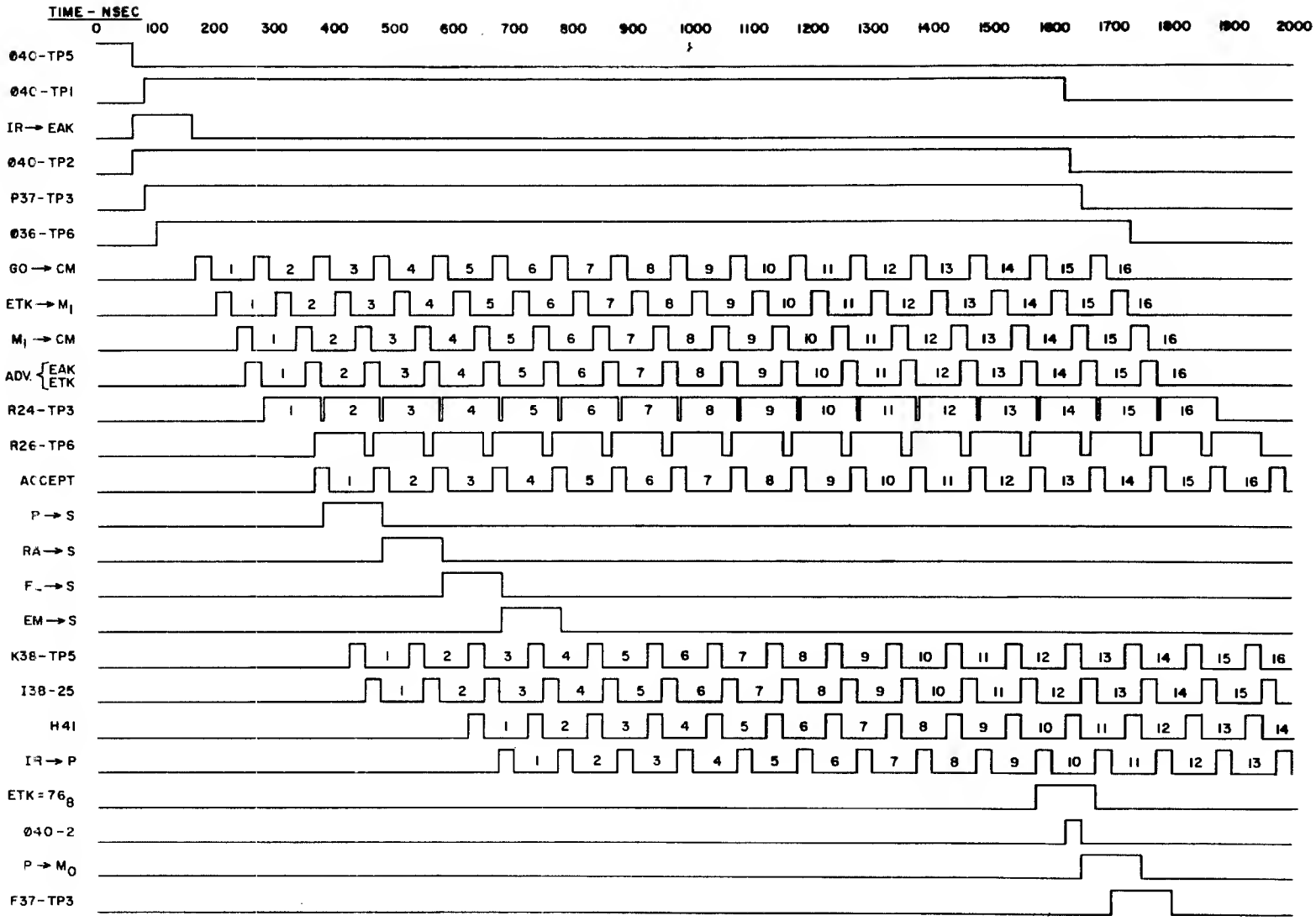


Figure 2-9. Exchange Jump Control

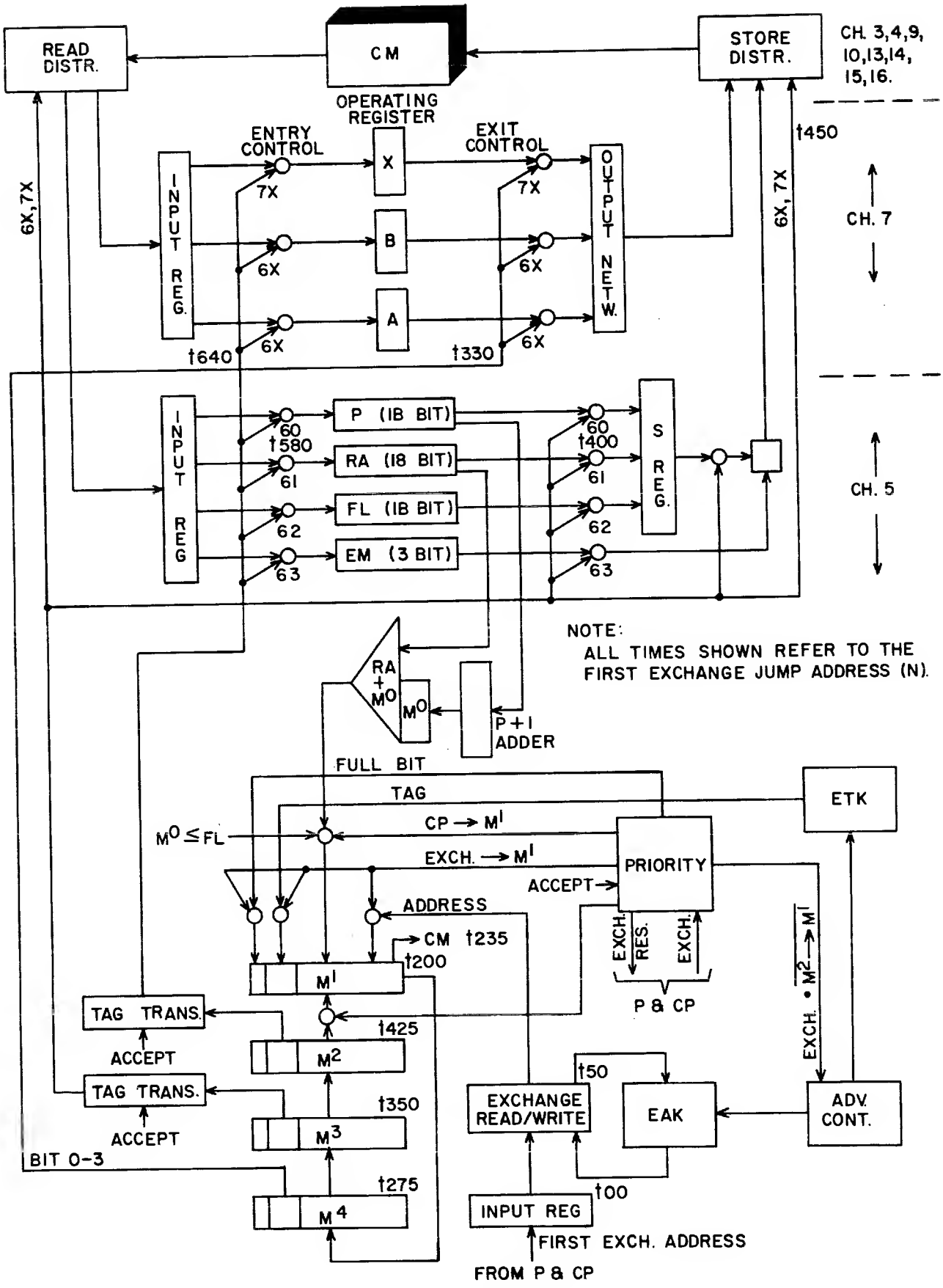
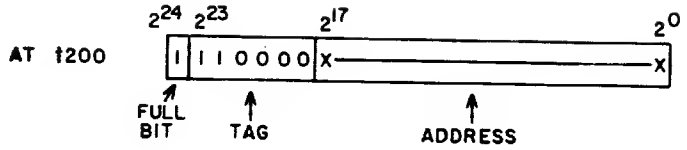


Figure 2-10A. Exchange Jump

following bits:



At t235, the address is sent from M^1 to all banks of central memory. As the largest possible address in central memory can be specified by 17 bits, bit position 2^{17} in M^1 is not sent to central memory. At t275, all 25 bits of M^1 start their cycle through the Hopper (M^1 to M^4 to M^3 to M^2). Once the address has been accepted by central memory, an "accept" signal is sent back (130 ns after sending M^1 to central memory) preventing the address from re-entering M^1. From M^4, M^3 and M^2 the tag bits are extracted and sent to the Exit control of the Operating Registers and to the Tag Translators.

This complete the process of sending the first address of the Exchange Jump Package from the Stunt Box to central memory. Since the Exchange Jump Package occupies 16 sequential addresses in central memory, the next 15 addresses must be generated in the central processor. The Exchange Address Counter increments the first address and subsequent addresses to provide these fifteen sequential addresses. From the Exchange Address Counter, they will be sent to central memory at a maximum rate of one every 100 nanoseconds.

The Exchange Address Counter is advanced as follows:

> At t160, Q19/TP6 was set and remains set throughout the Exchange Jump. The "1" from Q19/P26 provides a "1" from "K" on R14 (t200) which advances the counter. The B gates on Q27, 28, 29 were enabled at t115 and also remain enabled throughout the Exchange Jump. At t250, the new address (N+1) enters the Exchange/Read/Write register.

At the same time, the Exchange Tag Counter increments the tag by one and sends it to M^1 along with the new address. When the Exchange Tag Counter reaches $76_{(8)}$ a "0" into 040/P6 sends an Exchange Resume (t1630) back to the Peripheral & Control Processor and also clears FF TP3 and FF TP4 (t1630). This drops the advance pulses for the Exchange Address and Exchange Tag Counters. However, the counters still have time to reach $N+17_8$ and 17_8 respectively, completing the Exchange Jump in the Stunt Box.

OPERATION

The previous section described the controls for an Exchange Jump. This section attempts to explain the sequential operation of the exchange jump in the central processor.

The purpose of an exchange jump is to exchange the controls and parameters for one program with new controls and parameters for a second program. In the 6600 central processor the controls and parameters for any program presently running will be found in the 24 operating registers A, X, B and in the control registers P (program address), RA (reference address), FL (field length) and EM (exit mode). Before the exchange jump was executed, values for the above listed registers were loaded into an area of central memory and called the exchange jump package. The starting point of this area in memory was defined by the address N when the exchange jump was executed.

When the first address of the exchange jump entered the hopper, a tag was added. This tag, when translated, provides the gates necessary to exchange the P register and the A0 register with the contents of address N of the exchange jump package.

P ⟶ Address N

A tag translation of 60 and an "accept" signal enables a "0" from P40/P1 at t400, which goes to the S register to enable the contents of the P register into the S register. Then at t465, the contents of P are sent to the Store Distributor. From M^3, the 6-bit tag is extracted (t380) and transferred to the tag translator, P40. Together with an accept it will provide a "0" out of P40/P20 (6X and Accept) which is fed through 041/P26 to K38/P20. At t450 a Central Register Write signal is sent to the Store Distributor allowing the P register to be written into address N, bit positions 36-53, of central memory.

Address N ⟶ P

During the same memory cycle, the contents of P stored in the Exchange Jump Package is read from address N and sent via the Read Distributor to the Input Register (B42, A41,A42, C41 on Ch. 5). A 60 tag from P39/P1 and an "accept" enables the contents of the Input Register into the P register at t580.

AØ ⟶ Address N

The lower 4 bits of the 60 tag are extracted from M^4 and sent to GO2 (Ch. 8) and H26 (Ch. 7) of the Exit Control. The 2^3 bit (the "GO" bit) is a "0" which indicates that an A or B register can be

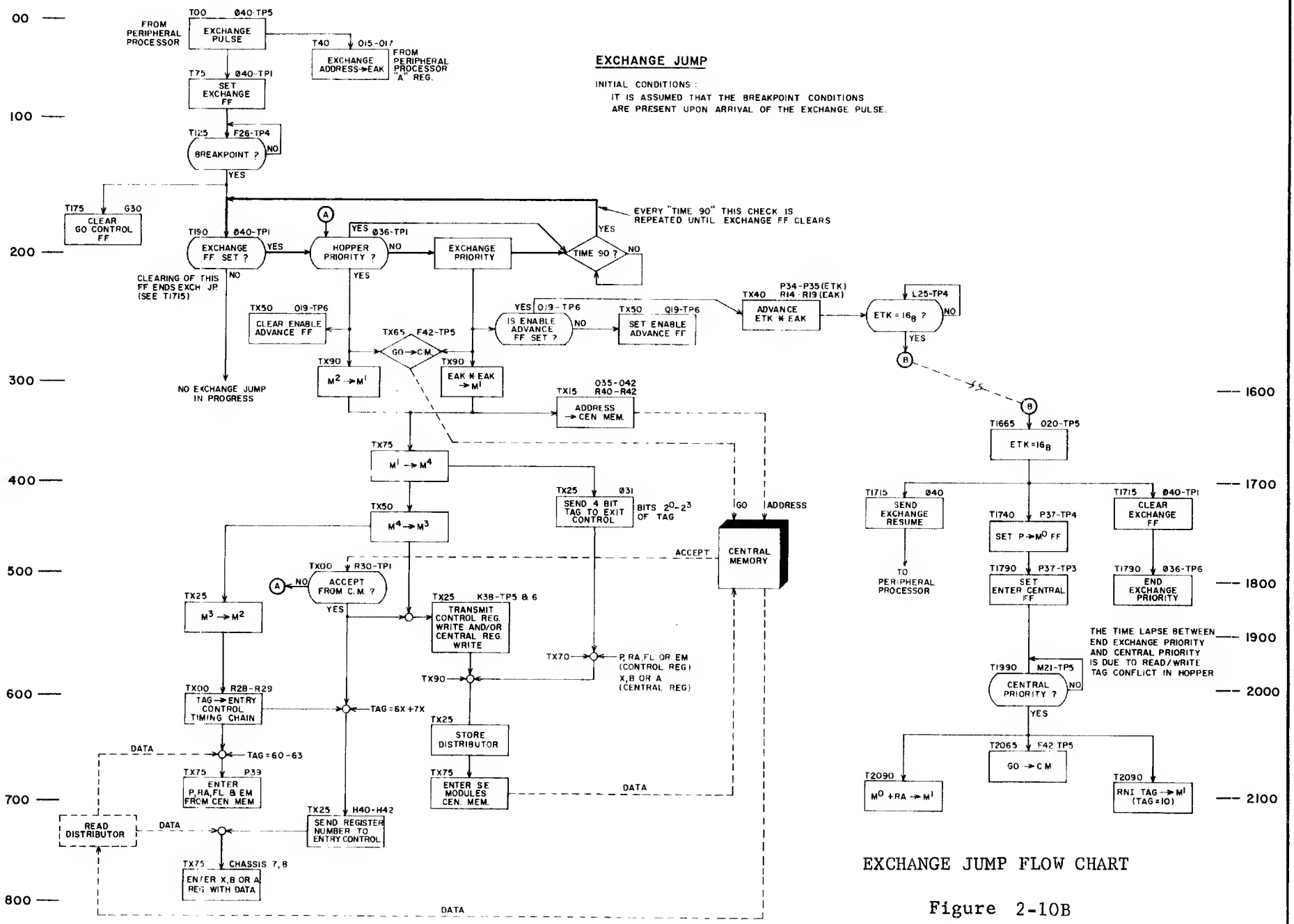
transferred into memory. The lower 3 bits are also "0"'s, indicating the register number is 0. As the BØ register is not connected to memory, we can only send the AØ register to memory. The "1" from H26/P3 (Ch. 7) is AND'ed with t30 on E31 to give a "0" output from E31/P18. This "0" goes into F32, 33, 34, 35/P10 and E26, 27/P10 allowing the 18 bits of the A0 register to be sent to address N, bit positions 18-35, of central memory.

Address N $\longrightarrow$ AØ

From R29 thru P18 and 033 (Ch. 5) a G0 bit is sent to Chassis 7, G37/P15. This G0 bit passes thru C36, A35 and enables the C gates in the Input Register B40, 41, 42 etc. on Chassis 7. The output of P39/P27 on Ch. 5 (a "0") is fed through H42/P10 into Chassis 7, H27/P11, setting the G0 FF. This provides a "0" output from H27/12 thru C20, E28 and into C27/P16. This G0 bit enables the transfer of the Input register thru C27, 28 etc. into the AØ register G15, 16 etc.

This concludes the exchange of P and A0 (address N). As the second address, N+1, is sent to memory, the tags will be set to 61, enabling the exchange of RA, A1 and B1. When the Exchange Tag Counter reaches 76_8 the Enter Central FF on P37/TP3 is set. This forces a 10 tag (RNI) and a full bit into M^1 after all exchange jump tags (they have the "Write" bit set) have been accepted.

After all the 16 addresses have exchanged their contents, the new program address in P will be sent to M^1 together with the 10 tag and full bit to start the execution of the new program.



EXCHANGE JUMP FLOW CHART

Figure 2-10B

PERIPHERAL READ/WRITE

When a peripheral processor requests access to central memory for a read or write, the peripheral processor sends the appropriate signal to the stunt box. The signal is received at time 00 on chassis 5, module 041. The peripheral processor also sends the central memory address from its "A" register. The address is received in the input register in chassis 5 at time 00 on modules N40, N41, N42. The appropriate read or write signal received at 041 generates a 50 nanoseconds one-shot pulse on pins 2, 4, and 8, to gate the address from the input register to the Exchange/Read/Write register located on modules Q15, Q16 and Q17. (Refer to Figure 2-11)

At this time, the controls on 041 attempt to gain priority to enter the hopper (signal on pin 10). Since priority to enter the hopper may not necessarily be available, the address in the exchange register is retained until it can be transferred to the hopper. To accomplish this, the address circulates through the exchange address counter (EAK) located on modules R14 through R19. The address is not advanced, however, since the advance pulse is only enabled during an exchange jump. (Refer to Figure 2-12)

When the peripheral address is attempting to gain priority in the stunt box, the "Central Busy" flip-flop in the peripheral processor is set. This prevents any other peripheral processor from attempting a central memory reference. The "Central Busy" flip-flop will only be cleared after the last address has been accepted by central memory.

To avoid the situation where the central processor ties up the hopper such that a peripheral processor request for memory cannot be honored, the circuitry stops the central processor, honors the peripheral request, and then restarts the central processor. Conditions for stopping the central processor are: (See Figure 2-13)

1) Address in M^2 not accepted.
2) M^1 full
3) M^3 full
4) M^4 full
5) Peripheral read or write request.

FULL BITS AND TAGS

When priority for the peripheral read or write address has been established, the priority network enables the gating of the address into the hopper (gates on Q25, Q26 and Q27). At this time a tag and Full bit will be appended to the address.

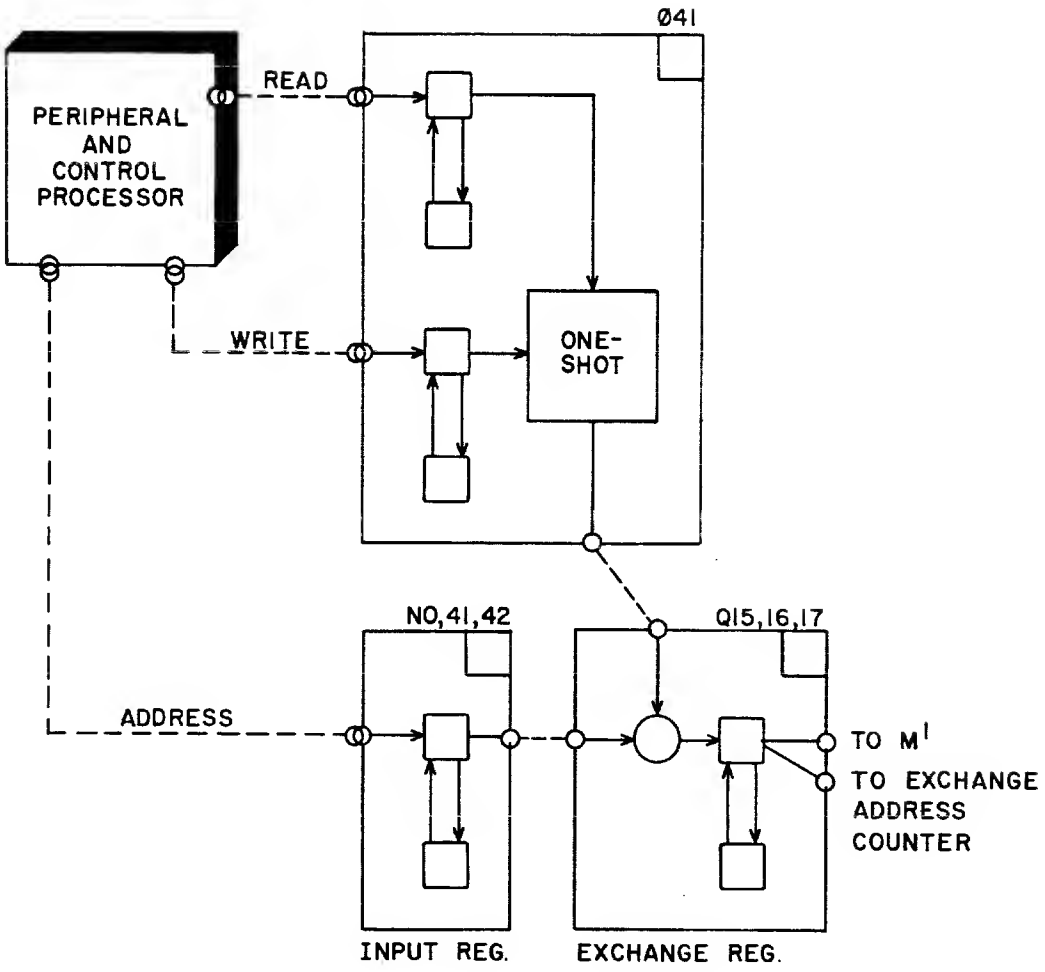

Ø41
PERIPHERAL AND CONTROL PROCESSOR
READ
WRITE
ONE-SHOT
NO,41,42
Q15,16,17
ADDRESS
TO M'
TO EXCHANGE ADDRESS COUNTER
INPUT REG.
EXCHANGE REG.

Figure 2-11

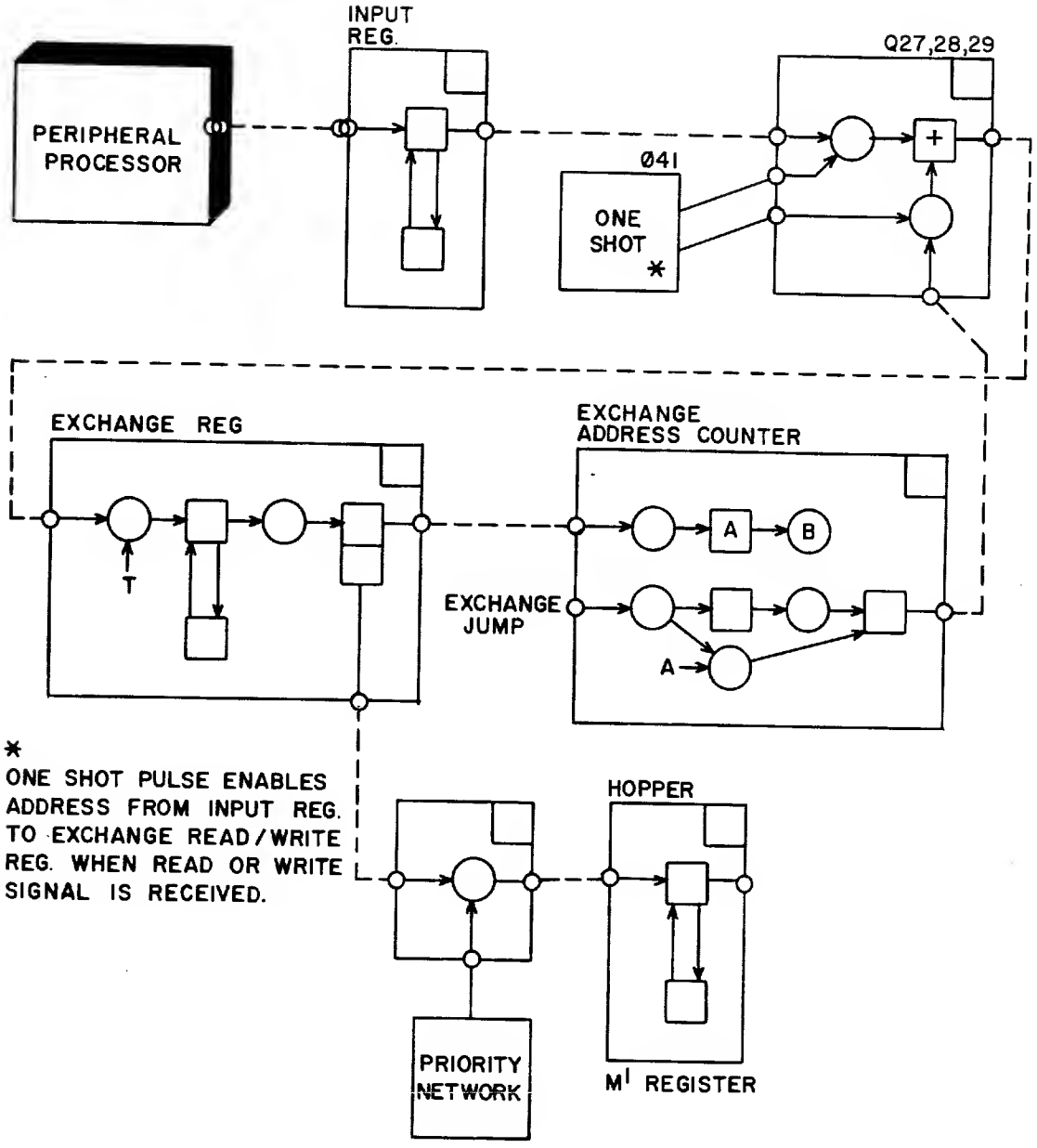

INPUT
REG.
PERIPHERAL
PROCESSOR
Q27,28,29
Ø41
ONE
SHOT
*
EXCHANGE REG
EXCHANGE
ADDRESS COUNTER
A
B
T
EXCHANGE
JUMP
A
*
ONE SHOT PULSE ENABLES ADDRESS FROM INPUT REG. TO EXCHANGE READ/WRITE REG. WHEN READ OR WRITE SIGNAL IS RECEIVED.
HOPPER
PRIORITY
NETWORK
M' REGISTER

Figure 2-12

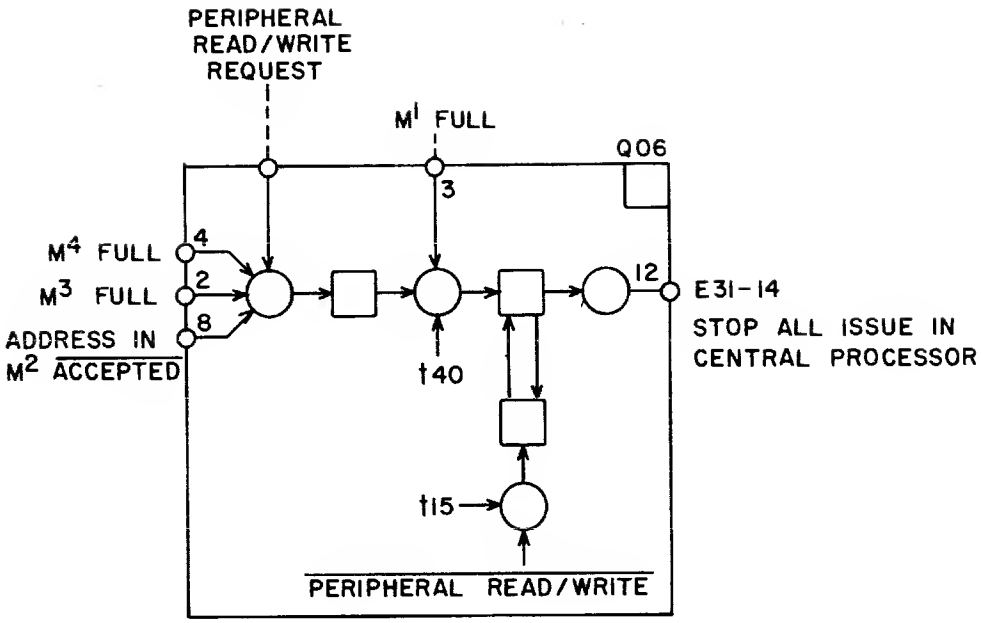


Figure 2-13

Full Bit

Addition of the Full bit is accomplished by setting flip-flop R33/TP1 with the "address to M^1" enable. (Refer to Figure 2-14)

Tag

The tag for a peripheral read is 00; the tag for a peripheral write is 40. One of these two tags must be appended to the address in the hopper when a peripheral read or write is attempted. When priority has been granted for gating the peripheral address to M^1, the "enable peripheral to M^1" term on 033/TP1 makes the "C" and "F" gates for the fan-in on Q25. This fan-in permits entry of the proper tag into M^1. In a peripheral read, all the inputs to the fan-in will be zero.

In a peripheral write operation the write bit (2^5) will be a "1" since R33/TP2 is set. Once the peripheral address is in the hopper, the address will be issued to central memory every 300 nanoseconds until it can be accepted. When the address has been accepted, the tag is translated from the hopper and sent to memory control to allow memory to send the data and resume signals to the peripheral processor.

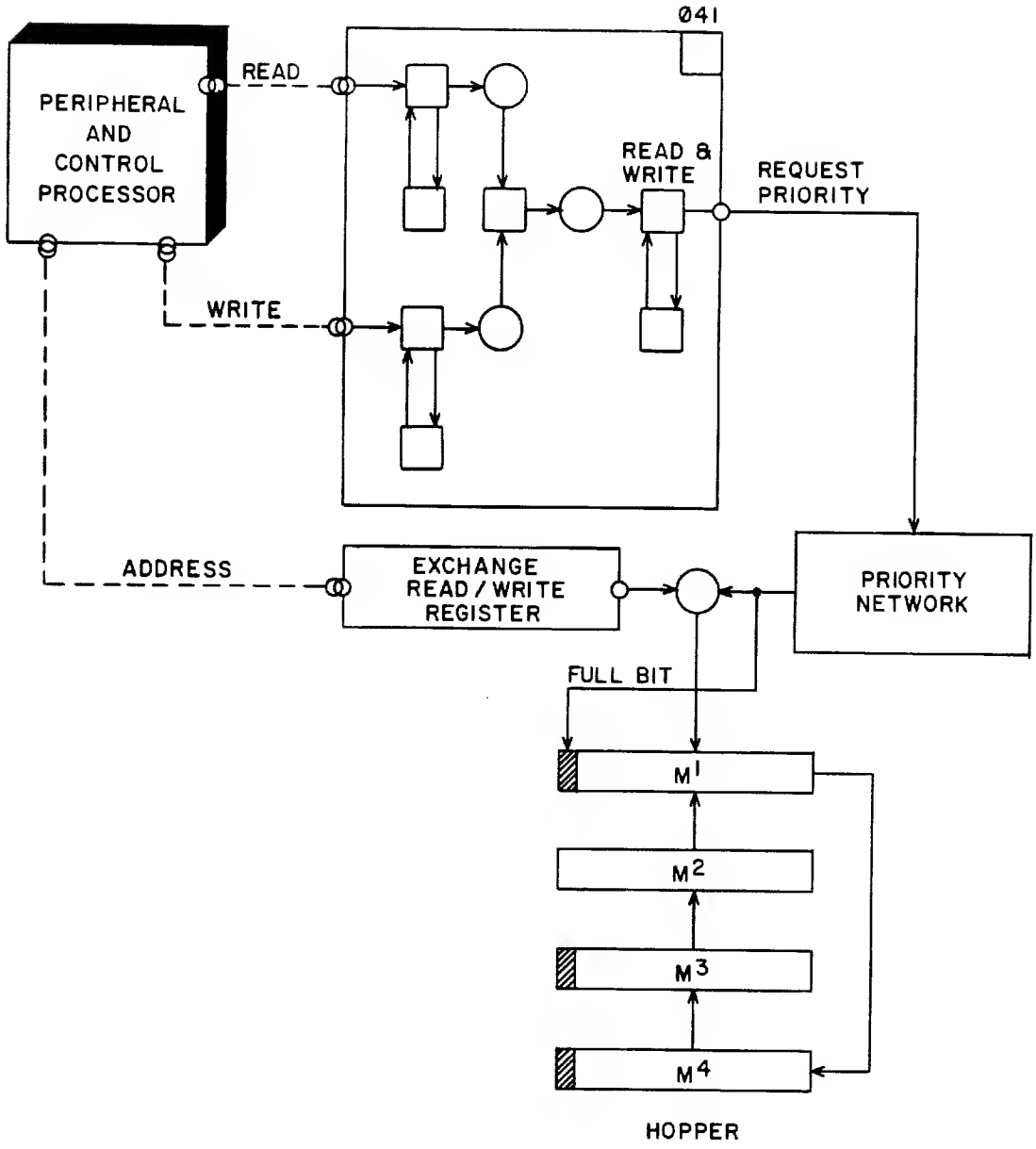

041
PERIPHERAL AND CONTROL PROCESSOR
READ
WRITE
READ & WRITE
REQUEST PRIORITY
ADDRESS
EXCHANGE READ / WRITE REGISTER
PRIORITY NETWORK
FULL BIT
M1
M2
M3
M4
HOPPER

Figure 2-14

CENTRAL READ/WRITE

When the central processor attempts a central memory read or write operation, the address originates from one of two basic sources:

1) from the Program Address (P) register when the memory request is for an instruction word, and

2) from one of the Increment functional units when the memory request is for an operand.

As diagrammed in Figure 2-15, either the Increment Address FF or the Program Address FF sets the Enter Central FF. Setting the Enter Central FF initiates action to request priority for the address to enter the hopper.

The address, in either case, is gated to a common register (M0). The contents of M0 are added to the reference address (RA) and this address is gated to hopper register M^1 when priority has been granted and $M_0 \geq FL$.

PROGRAM ADDRESS

The central processor Program Address register (P) and its incrementor network always hold the address of the last instruction word read from memory. When the next instruction word is needed, an "Inch" signal sets the Program Address FF. This gates the contents of the P register + 1 to the M_0 register, providing M_0 is not holding an address that has not yet entered the hopper. In addition, if there are addresses coming to M_0 simultaneously (i.e., from P and from the Increment units), the Increment address has priority over P+1 for entry into M0.

INCREMENT ADDRESS

When an instruction that causes a change in A registers one through seven (A1 - A7) is executed in the Increment units, the address developed is sent to the Stunt Box to make the necessary memory reference for loading or storing operating register X^1 - X^7.

An increment instruction (51 - 57), when nearing completion, sets the Increment Address FF. Subsequent events are as outlined above.

Figures 2-16 and 2-17 diagram the sequence of events for central memory read and write operations.

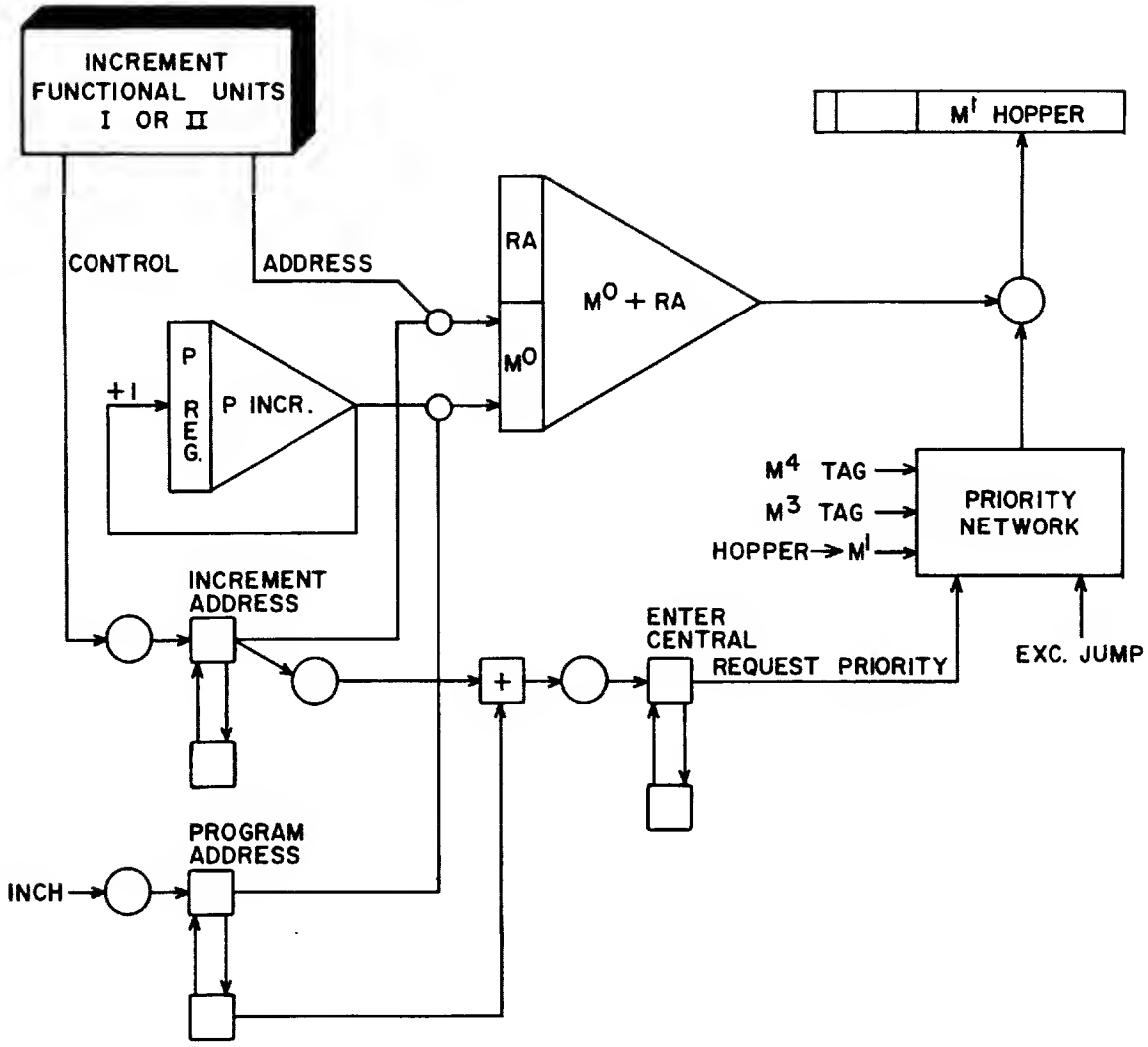
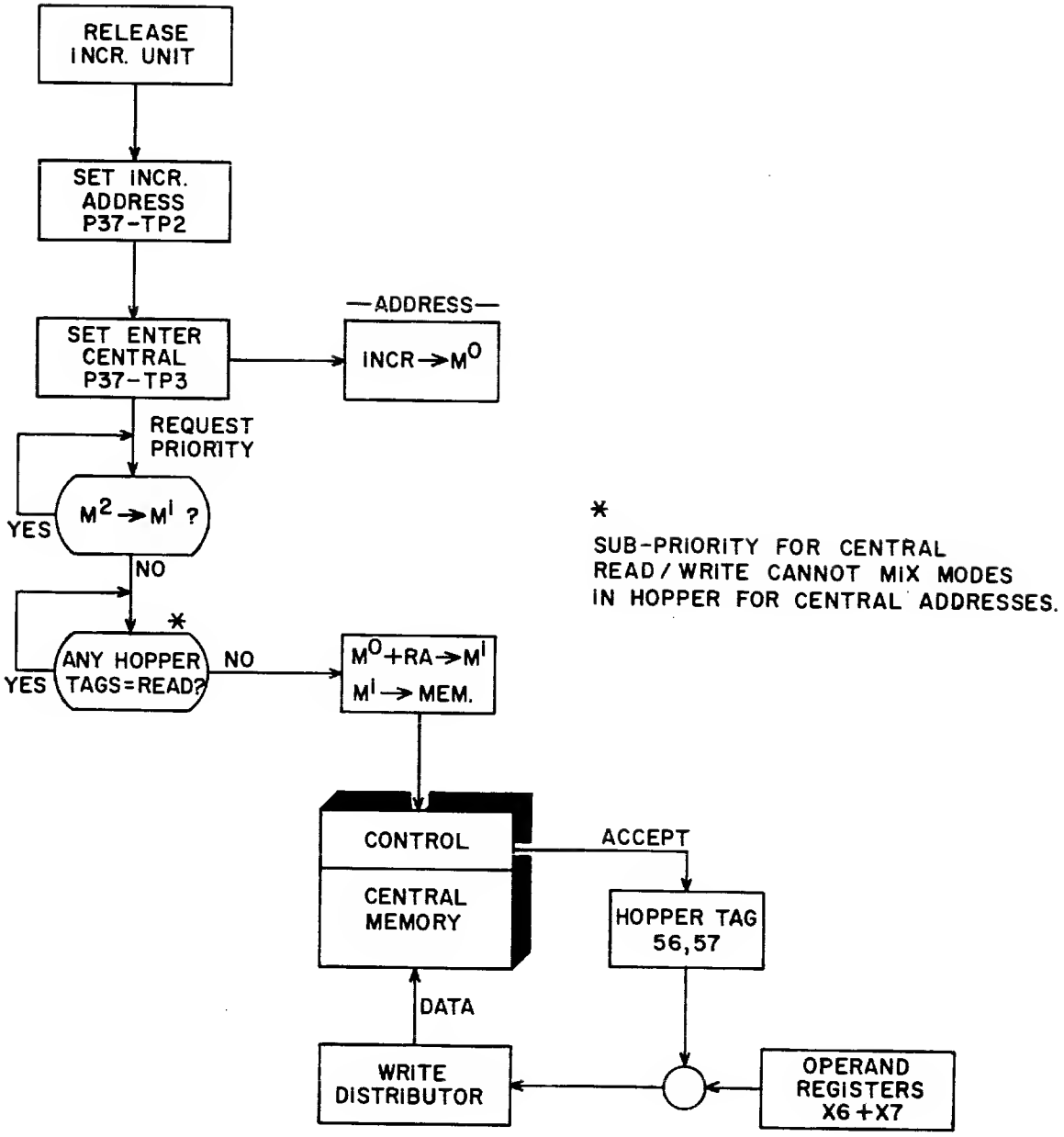
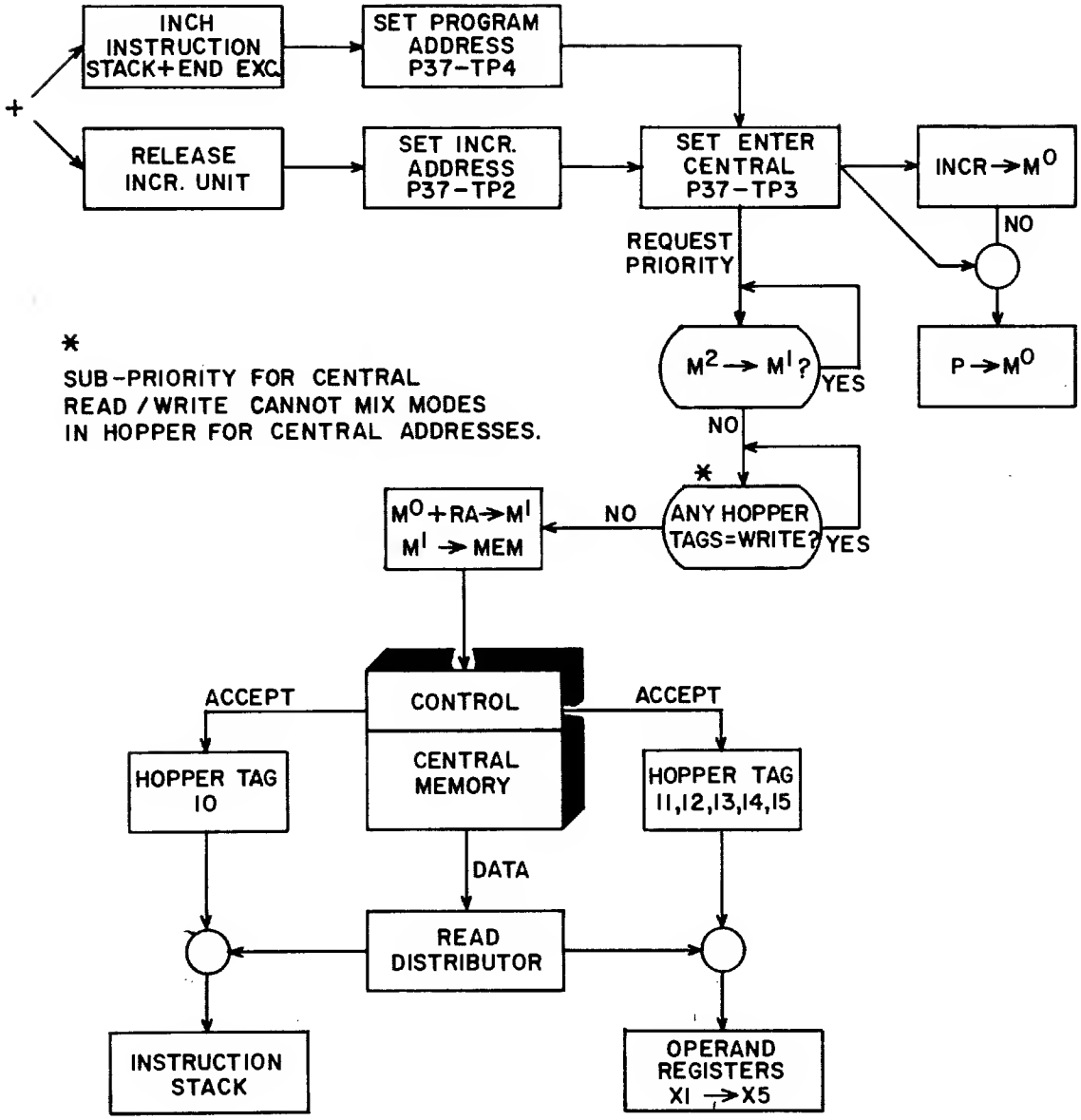


Figure 2-15. Central Read/Write



CENTRAL WRITE

Figure 2-16



CENTRAL READ

Figure 2-17

EXIT MODE

The exit mode feature allows the programmer to choose the exit or stop condition of the central processor. Exit selections are stored in the EM-register, and the exit mode occurs as soon as it is sensed. The various exit conditions are:

1. Normal stop
2. Address out of bounds (MO $\geq$ FL)
3. Operand out of range (infinite)
4. Indefinite operand

The address RA is reserved for recording program exit conditions.

NORMAL STOP

The translation of a normal stop instruction (fm = 00) in the U2 translator together with a SCBD ISSUE will set the STOP FF (G30 TP1). The P register contains the address or address + 1 of the STOP instruction. The peripheral and control processor searches for an unchanging central processor P register and upon finding it, searches that location for fm = 00. If found, the CP stopped normally; if not, it can be assumed that the program is looping.

ADDRESS OUT OF BOUNDS

MO $\geq$ FL test

The contents of the MO register are continuously compared with the contents of the FL register as follows:

a. A bit by bit comparison of bits $2^3 \longrightarrow 2^{17}$ (MO $\geq$ FL) and three results of these comparisons are comprised in one group (ANDed together). If one bit of MO is less than the corresponding bit of FL the group output is "0".

b. Each of the 6 octal digits of MO is compared with the corresponding octal digit of FL. If MO $\geq$ FL the output will be a "1".

The output of (a) and (b) are ANDed together to decide if MO $\geq$ FL. If MO $\geq$ FL and we are not in an Exchange Jump the MO $\geq$ FL FF (013 TP5) will be set. Supposing the contents of the EM register is XX1 and the Skip III FF is set, the MO $\geq$ FL will set the ERROR FF.

The output of the comparison network will also inhibit sending the write bit of the hopper tag to the hopper and does not allow the value of MO + RA to be sent to M1 if MO $\geq$ FL.

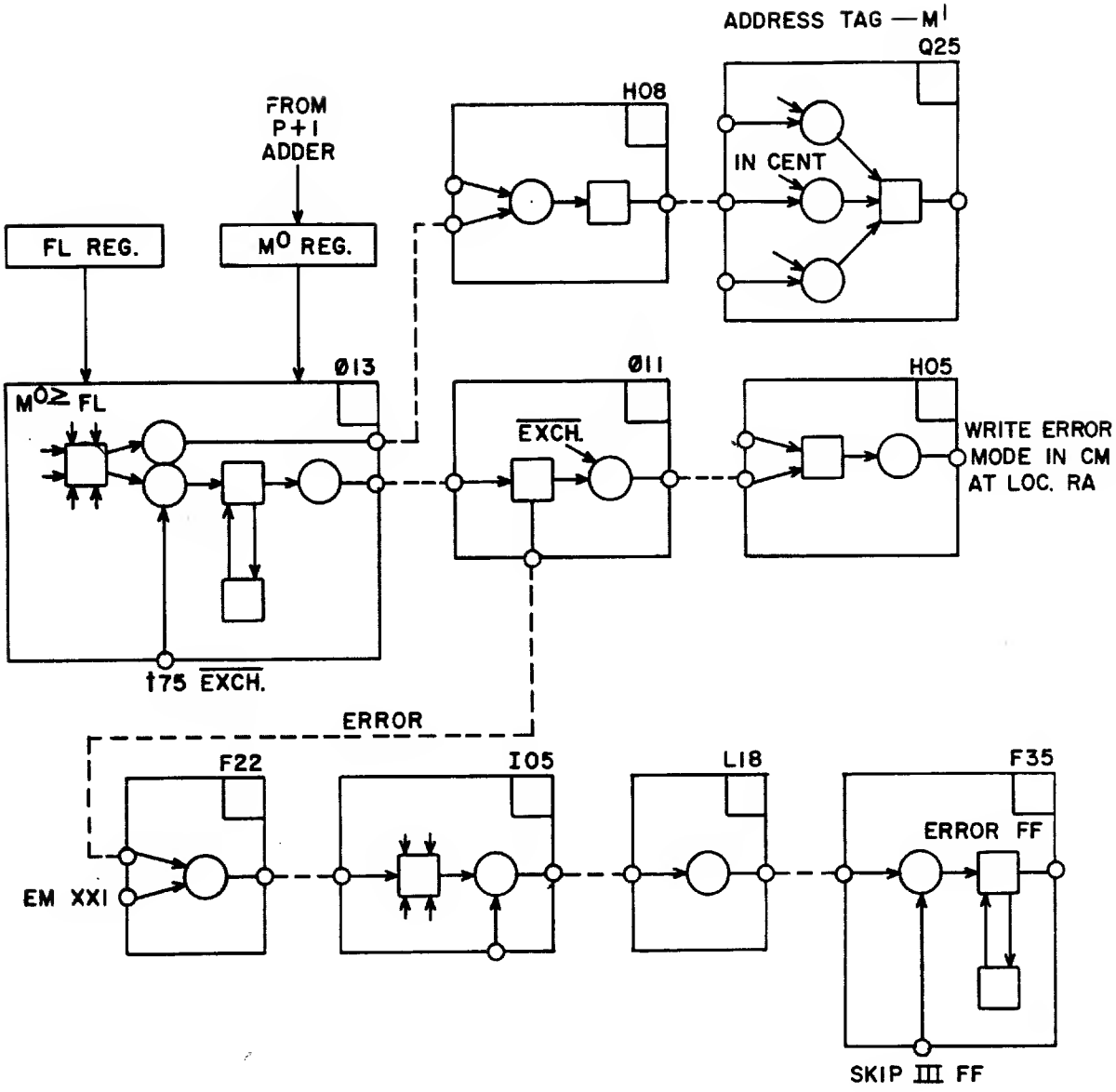

ADDRESS TAG — M1
Q25
H08
FROM
P+1
ADDER
IN CENT
FL REG.
M0 REG.
Ø13
Ø11
H05
M0≥ FL
EXCH.
WRITE ERROR
MODE IN CM
AT LOC. RA
↑75 EXCH.
ERROR
F22
I05
L18
F35
ERROR FF
EM XXI
SKIP III FF

Figure 2-18

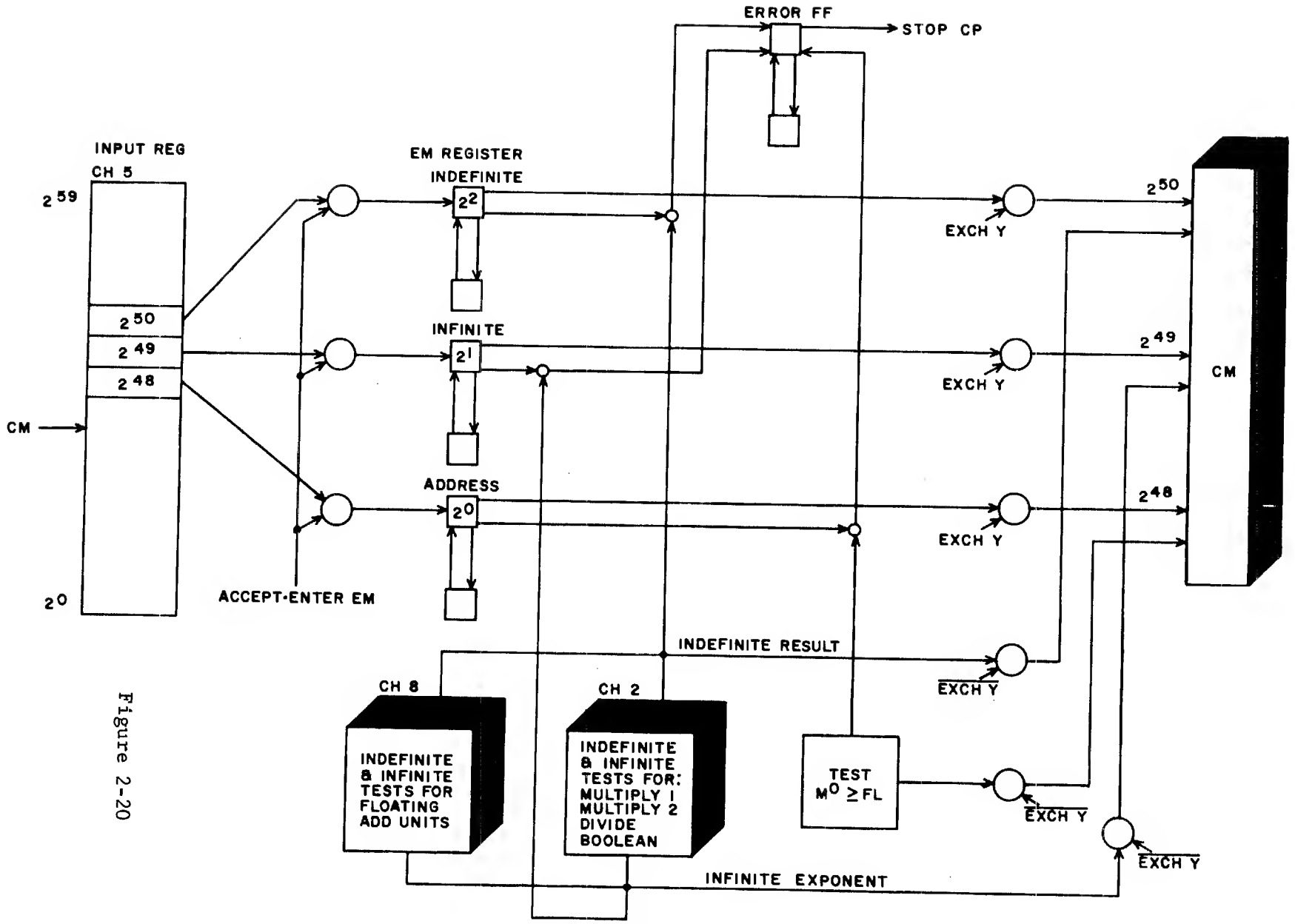


Figure 2-20

EXIT MODES

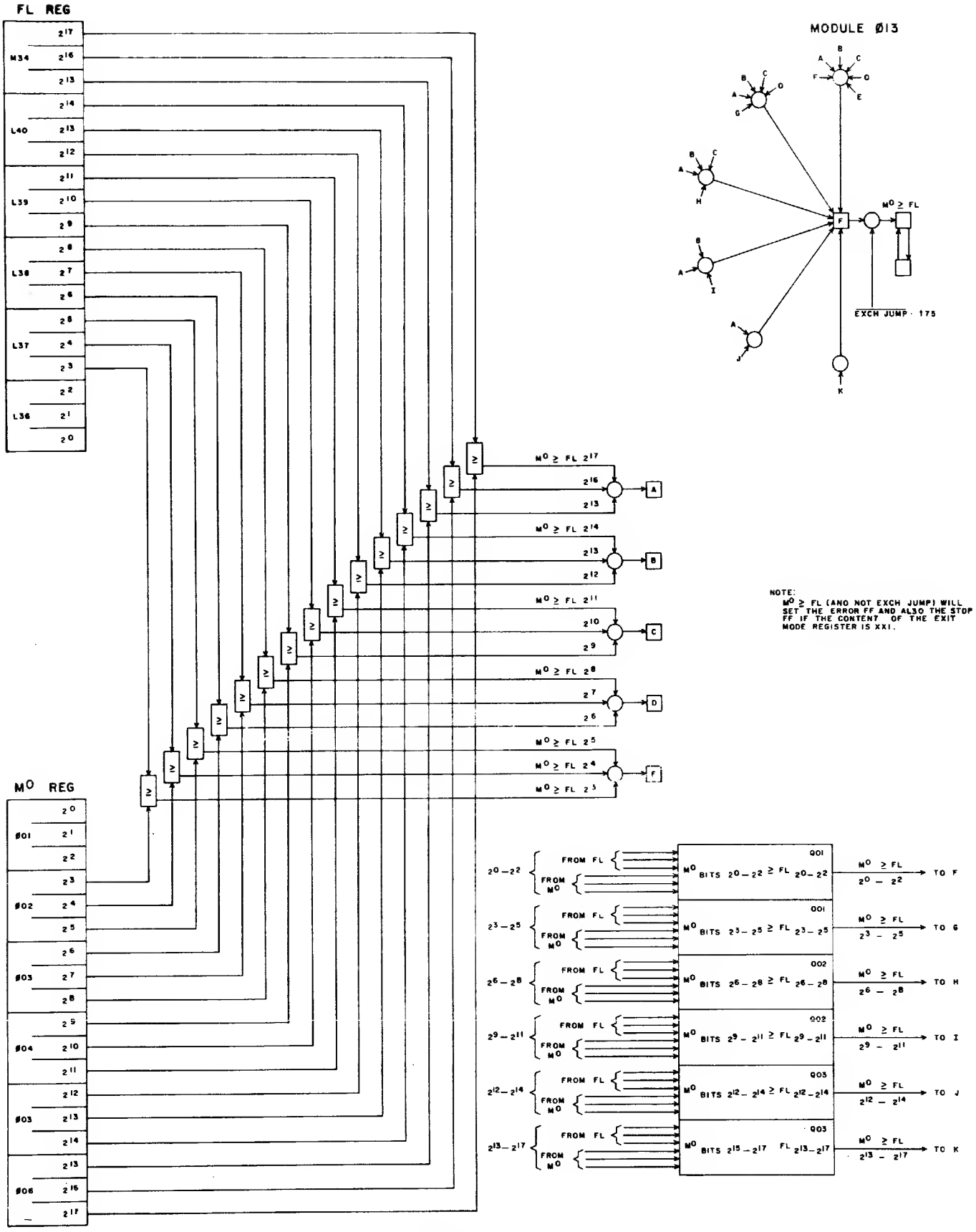


Figure 2-21

M° - FL Comparison

Address out of bounds, exit mode not selected (EM) = XX0

When the central processor attempts to reference a memory location out of bounds, an out-of-bounds exit mode is not selected. If out of bounds is selected, the central processor will stop at the Reference Address.

RNI Out of Bounds. If the program counter (P register) attempts to send an address that is out of bounds (this would be the RNI situation), the MO $\geq$ FL check circuit will inhibit sending the intended address to the hopper. Instead, an address of all zeros and the proper RNI tag will be sent to the hopper. This will result in the reading of memory address absolute zero. The data will go to the instruction stack and the U register translating networks. If absolute address zero contained an all zero word, the central processor would stop due to the translation of a stop (00) instruction.

Read Operand Out of Bounds. If the increment unit changes A1-A5 (read memory to X) with an address that is out of bounds, again the MO $\geq$ FL check circuits will inhibit the sending of the address to the hopper. Instead an all zeros address will be sent to the hopper with a 10 tag. Location absolute zero will be referenced, but its contents will be lost in the Data Distributor. (Since tag = 10 can't be translated in control). No affect on the running program will be seen except that the contents of the X registers will not be what was expected.

Write Operand Out of Bounds. If the increment unit changes A6 or A7 (store X6 or X7) with an address that is out of bounds, the MO $\geq$ FL check circuit will inhibit the sending of the out bounds address to the hopper. Instead an address of all zeros will be sent to the hopper. Usually the tag for a store X6 or X7 is a 56 or 57, but if the address is out of bounds the tag will be changed to a 16 or 17, this tag will inhibit sending the central register write tag to the store distributor. The results of this zero address and 16 or 17 tag in the hopper is the referencing of address absolute zero. When the data gets to the data distributor however, the lack of any tag will cause the data to be lost. There will be no effect on the running program, except that the desired content of the X6 or X7 register would not be stored.

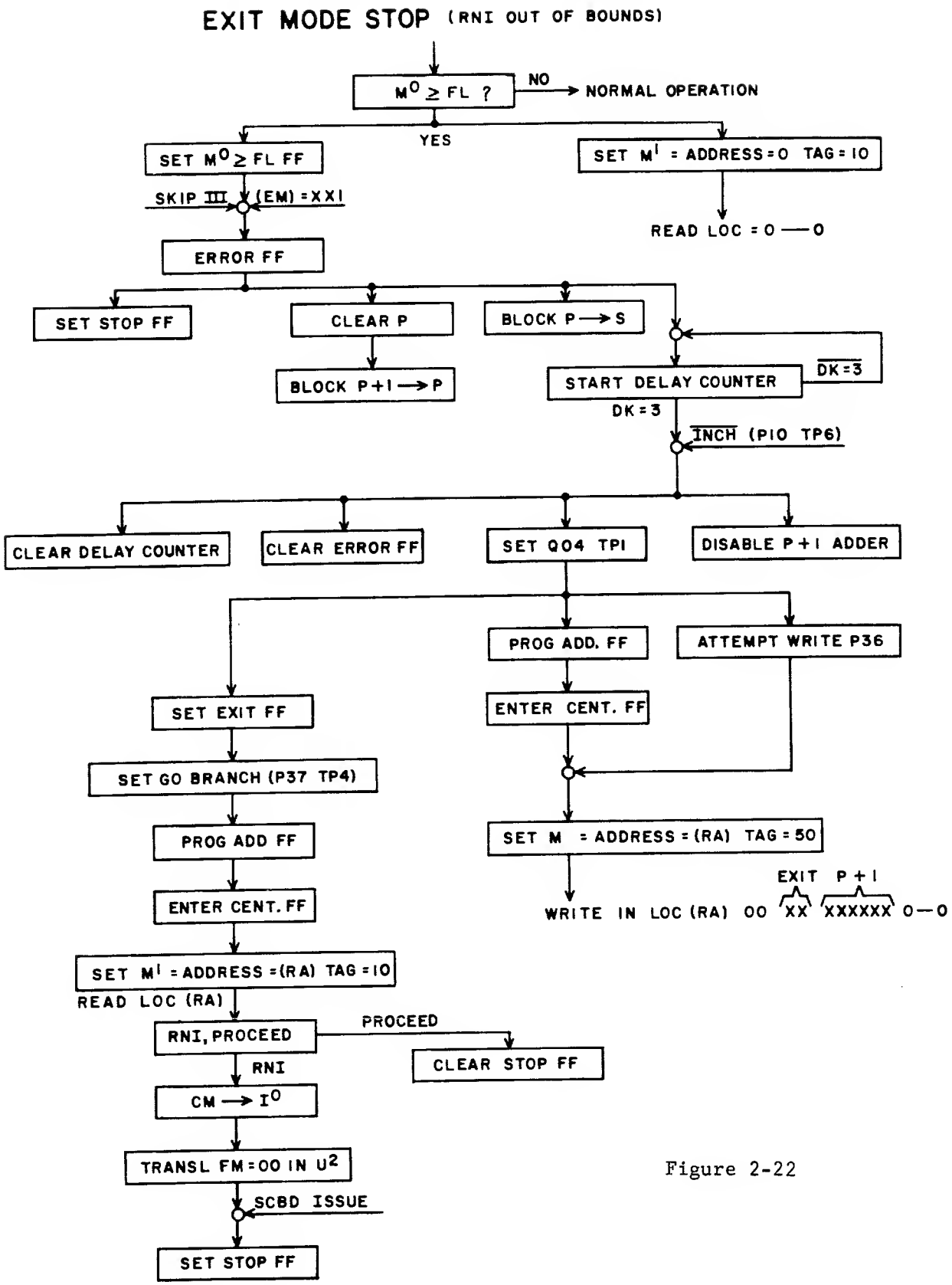

EXIT MODE STOP (RNI OUT OF BOUNDS)
M0 ≥ FL ?
NO
NORMAL OPERATION
YES
SET M0 ≥ FL FF
SET M1 = ADDRESS = 0 TAG = 10
SKIP III (EM) = XX1
READ LOC = 0 — 0
ERROR FF
SET STOP FF
CLEAR P
BLOCK P → S
BLOCK P+1 → P
START DELAY COUNTER
DK = 3
DK = 3
INCH (P10 TP6)
CLEAR DELAY COUNTER
CLEAR ERROR FF
SET Q04 TP1
DISABLE P+1 ADDER
PROG ADD. FF
ATTEMPT WRITE P36
SET EXIT FF
ENTER CENT. FF
SET GO BRANCH (P37 TP4)
PROG ADD FF
SET M = ADDRESS = (RA) TAG = 50
EXIT P+1
WRITE IN LOC (RA) 00 XX XXXXXX 0 — 0
ENTER CENT. FF
SET M1 = ADDRESS = (RA) TAG = 10
READ LOC (RA)
RNI, PROCEED
PROCEED
CLEAR STOP FF
RNI
CM → I0
TRANSL FM = 00 IN U2
SCBD ISSUE
SET STOP FF

Figure 2-22

Address out of bounds, exit mode selected (EM) = XX1

(Refer to timing diagram of Exit Mode Stop)

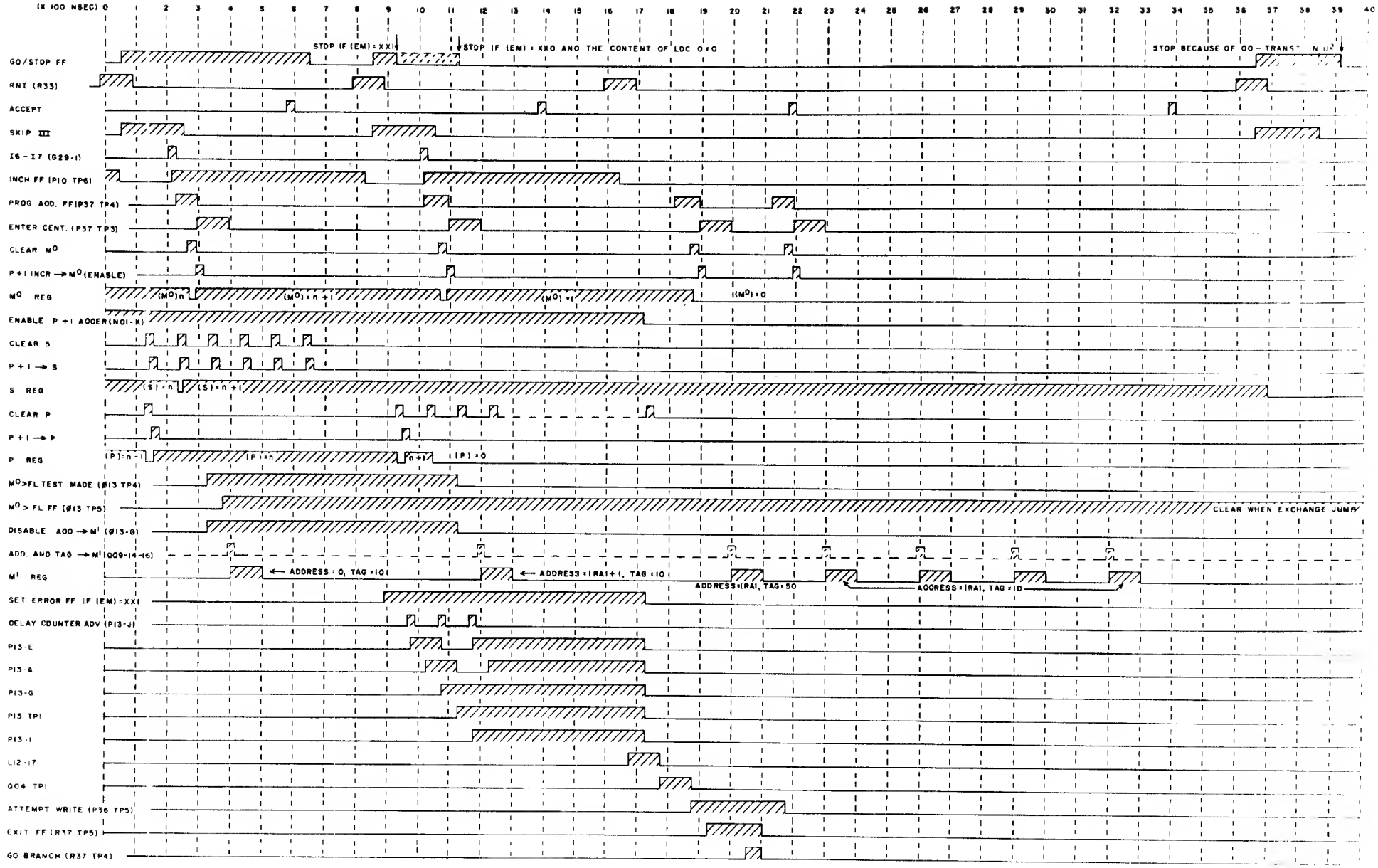
If the M0 $\geq$ FL comparison network detects an address out of bounds it will inhibit the sending of the intended address to the hopper and will also set the M0 $\geq$ FL FF. Instead, an address of all zeros and a 10 tag will be sent to the hopper. A 10 tag in the hopper tag timing chain and an ACCEPT signal will set the RNI FF and a PROCEED signal is sent to the ISSUE CONTROL. As soon as the SKIP III FF is set the ERROR FF is also set. This in turn will start the DELAY COUNTER (P13) and also set the STOP FF. After 200 nsecs. the output of the counter is 1, but the INCH FF (P10 TP6) must be cleared in order that the Delay counter can clear the FF at Q04-TP1. This FF will enable the setting of the PROGRAM ADDRESS FF and with a P $\longrightarrow$ M° signal will also set the ATTEMPT WRITE FF. Now the address RA and a 50 tag will be sent to the hopper (M1) and the following data will be written into CM at location RA:

Stop	Exit	(P) + 1	
00	XX	XXXXXX	0 $\longrightarrow$ 0

At the same time the ATTEMPT WRITE FF and the FF at P10 TP1 was set. The latter will set the EXIT FF and this will set the GO BRANCH FF allowing a jump to P = 0 and hence RA. 300 ns after the address RA and a 50 tag was sent to the hopper the same address with a 10 tag will be sent to M^1. When the address is accepted by CM the contents of address RA (now 00 XX XXXXXX 0000000000) will be transferred to the Instruction Stack and to the U registers (because of the RNI and Proceed signals). The translation of fm = 0X in the U^2 translator and a SCBD Issue will set the STOP FF again.

INFINITE OR INDEFINITE OPERANDS

The functional units using floating point arithmetic will always test the exponents for infinity (greater than or equal to 3777) and for indefinite results. (see Appendix B). If any one of the above conditions exists, a bit is sent to Chassis 5, where it is ANDed with the respective FF's of the EXIT MODE register to enable setting the ERROR FF. As soon as the ERROR FF is set the timing diagram of Exit Mode Stop for RNI out of bounds (Figure 2-23) can be used.



NOTE:
I = CONTENTS OF P REG AT TIME OF ERROR (P≥FL)

Figure 2-23 EXIT MODE STOP (RNI OUT OF BOUNDS)

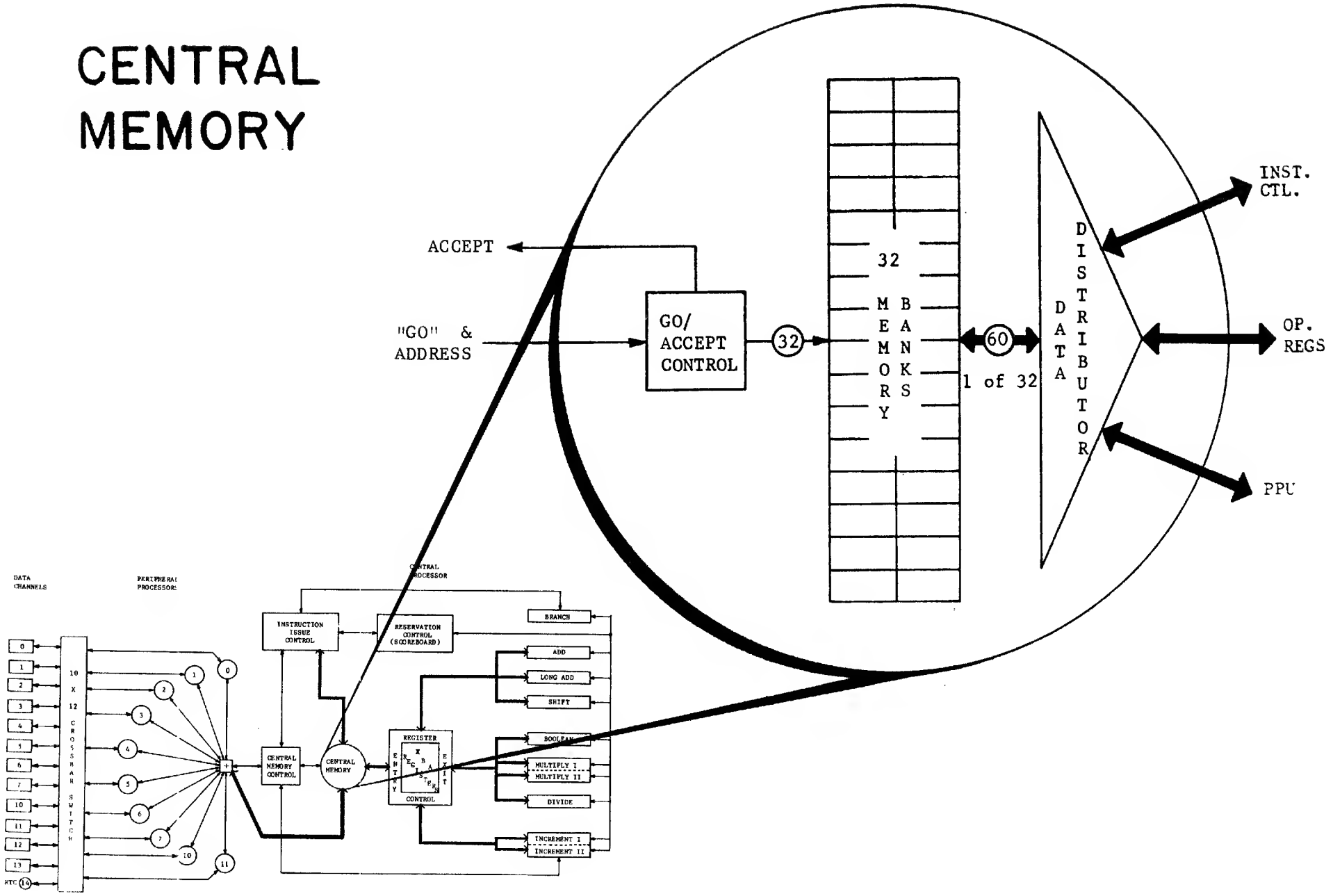
EXIT P+1

(WRITE IN LOC RA : OD XX XXXXX 0——0)

CHAPTER III

CENTRAL MEMORY CONTROL

CENTRAL MEMORY



CHAPTER III

CENTRAL MEMORY CONTROL

INTRODUCTION

The Central Memory (CM) can be accessed by the Central Processor and all peripheral processors at a maximum rate of once every 100 ns. This is the maximum issue rate of the Stunt Box, which collects and accept addresses from several sources on a priority basis.

The duration of the memory cycle for each memory reference is 1000 ns, or one major cycle, and consists of a read and write cycle which is controlled by the Storage Sequence Control circuitry.

During a Central Memory Read, the contents of the specified address are read out of memory and all cores in this location switched to "0". The contents are then routed to the Data Distributor and the Restoration Register. The Restoration Register allows data that has been read to be put back into memory during the write portion of the cycle. For a Write operation, the current contents of the specified address is read out of memory, but is destroyed in the Data Distributor. This data is not sent to the Restoration Register. Instead, the new data to be stored is accepted from the Data Distributor and is written into memory during the write part of the cycle.

Central Memory has 32 banks located on 8 chassis, 4 banks to a chassis. (A 65K CM has only 16 banks.) (Figure 3-2)

Address and Tag Format from Stunt Box follows:

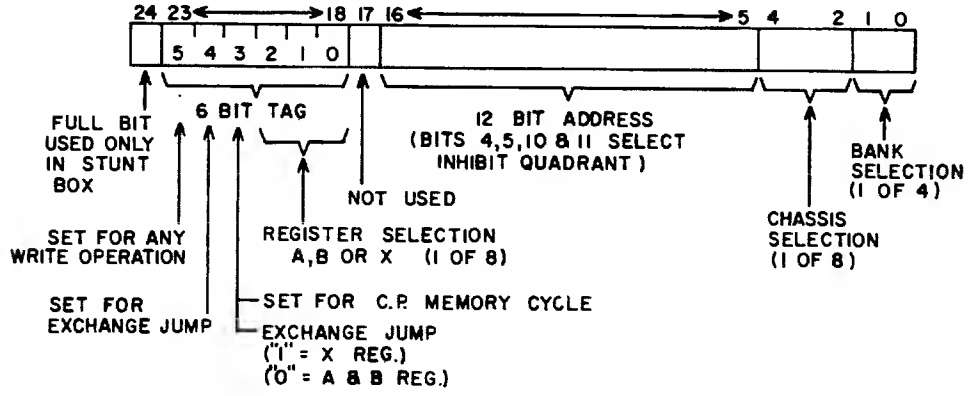


Figure 3-1

Translation Table for Bits 0-5 (131K Memory)

000 00	Chassis 3	Bank 0	Bank No. 1
000 01	Chassis 3	Bank 1	Bank No. 2
000 10	Chassis 3	Bank 2	Bank No. 3
000 11	Chassis 3	Bank 3	Bank No. 4
001 00	Chassis 4	Bank 0	Bank No. 5
001 01	Chassis 4	Bank 1	Bank No. 6
001 10	Chassis 4	Bank 2	Bank No. 7
001 11	Chassis 4	Bank 3	Bank No. 8
010 XX	Chassis 9	Bank 0-3	Bank Nos. 9-12
011 XX	Chassis 10	Bank 0-3	Bank Nos. 13-16
100 XX	Chassis 13	Bank 0-3	Bank Nos. 17-20
101 XX	Chassis 14	Bank 0-3	Bank Nos. 21-24
110 XX	Chassis 15	Bank 0-3	Bank Nos. 25-28
111 XX	Chassis 16	Bank 0-3	Bank Nos. 29-32

eg

M1 = 14272116 = Cent. Proc. Read $\longrightarrow$ X^4 Register
Address 5642 of Bank 2 in Chassis 10 Bank No. 15

Figure 3-2

GO/ACCEPT CONTROL

For every Memory reference requested, a GO signal along with the address is sent to all 32 banks of CM. from the M1 Register in the Stunt Box. If memory can be accessed (no bank conflict), an ACCEPT signal is sent back to the Stunt Box. See Figures 3-1 and 3-2 for 24-bit M1 format and the central memory chassis and bank distribution.

GO CONTROL (Figure 3-3)

The Go Control circuitry is located on Chassis 4, the GO being sent to Module I20, from where it is fanned out to all CM. Chassis. Bits 0 to 5 are sent simultaneously with the GO to the eight CM. Bits 2, 3, 4 are used for Chassis selection and corresponding to FF's B, C and D in G4/G38 (EVEN/ODD CH.). FF E is for the GO, and FF A is not used - a possible provision for a larger memory (8 more Chassis). FF's B, C and D are preset by a timing pulse to the complement of the Chassis number in which they are located. Bits 2,

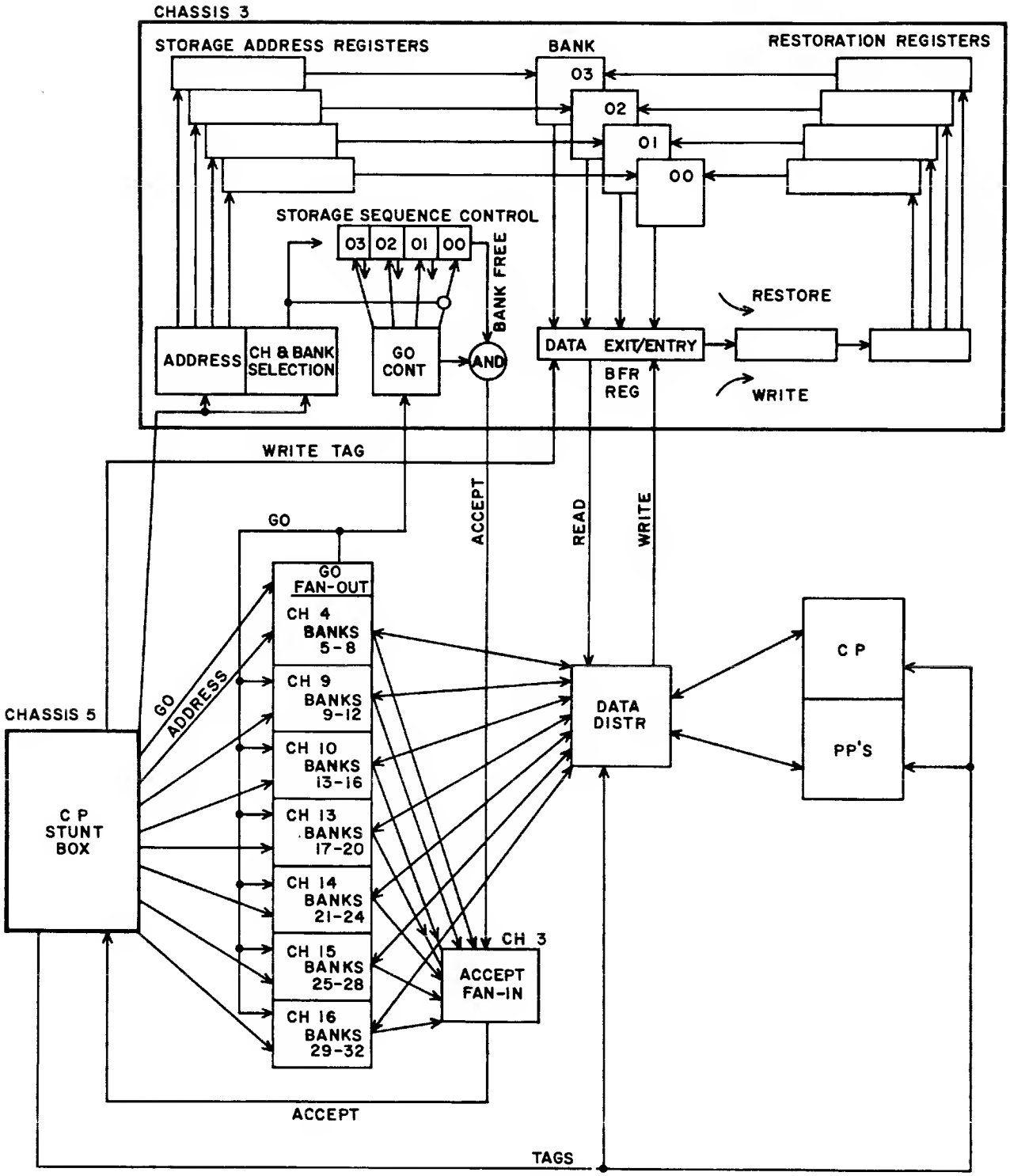

CHASSIS 3
STORAGE ADDRESS REGISTERS
BANK
RESTORATION REGISTERS
03
02
01
00
STORAGE SEQUENCE CONTROL
03 02 01 00
BANK FREE
RESTORE
ADDRESS
CH & BANK SELECTION
GO CONT
AND
DATA EXIT/ENTRY
BFR REG
WRITE
WRITE TAG
GO
ACCEPT
READ
WRITE
GO FAN-OUT
CH 4 BANKS 5-8
CH 9 BANKS 9-12
CH 10 BANKS 13-16
CH 13 BANKS 17-20
CH 14 BANKS 21-24
CH 15 BANKS 25-28
CH 16 BANKS 29-32
CHASSIS 5
GO ADDRESS
C P STUNT BOX
DATA DISTR
C P
PP'S
CH 3
ACCEPT FAN-IN
ACCEPT
TAGS

Figure 3-3

3 and 4 are wired in from the hopper appropriately to the set or clear inputs of the FF's. The outputs of B, C, D and the GO are ANDED and the GO transmitted to G5/G39 (EVEN/ODD CH.) if the gate is made. Bank selection is made here with bits 0 and 1 in a similar manner. An additional gating term, a bank free condition which ensures that the selected bank is not in previously initiated memory cycle, gates the GO to the storage sequence control and sends back an ACCEPT. No ACCEPT is sent to the Stunt Box if the selected bank is executing a storage cycle from a previously issued address. In this case the Address is saved in the Hopper and reissued every 300 nsecs,until accepted. In the worst case of bank conflict, an address can remain in the Hopper for 2700 ns (Refer to discussion on Stunt Box).

ACCEPT CONTROL (Figure 3-3)

The Accept indicates a bank is free and has accepted the address into its Chassis input register. From time the GO is issued from the Stunt Box until the time the Accept is received on chassis 5, is an interval of 200 ns. The Accept signal is generated at G5/G39 (EVEN/ODD CH.) of the respective Chassis and is routed to Chassis 3I23 which is an OR Circuit (a fan-in to send back the Accept from any of the eight Chassis). Hence an Accept can be sent back to the Stunt Box every 100 ns assuming no bank conflicts.

The GO and ACCEPT CONTROL Circuitry does the following:

1. Recognizes an address from the Stunt Box and determines its bank location.

2. Sends an Accept if the address is valid and the bank is free.
3. Starts the 1000 ns. Storage cycle.

ADDRESS PATH (Figure 3-4 & 3-5)

The 12-bit address is sent to the eight memory Chassis (G1/G40 EVEN/ODD CH.) 50 ns after the GO. In each Chassis the address fans out to all four banks. The address is gated by a Bank FREE signal into the storage address registers of all free banks, but Read Drive is turned on for only one bank - the selected address. Here, each bit fans out to five storage modules of 12 planes each, to select one 60-bit word location (Figure 3-4). Bits 4, 5, 10 and 11 have an additional five outputs to select the inhibit quadrant in each storage module (Figure 3-5). The storage address register is statically translated during the whole storage cycle.

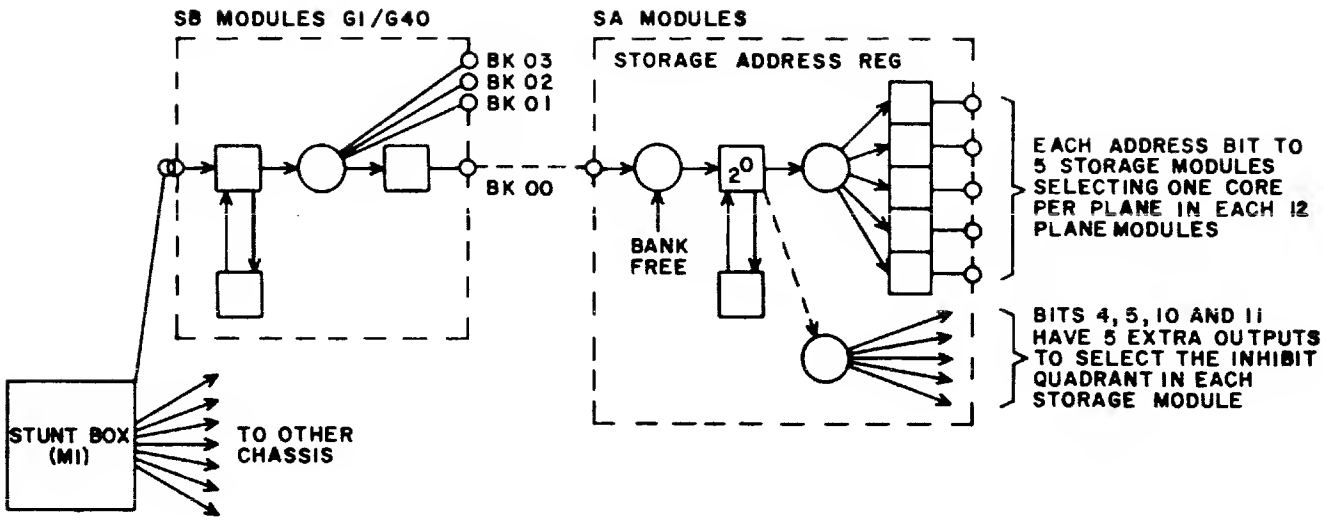


Figure 3-4. Address Path

FIG. 4-13 ADDRESS PATH

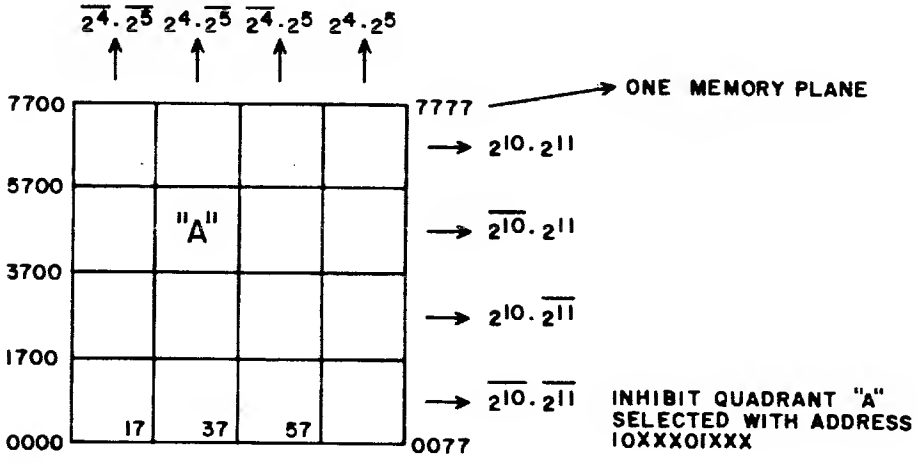


Figure 3-5. Inhibit Quadrant Selection

STORAGE SEQUENCE CONTROL
(Figures 3-6 & 3-7)

The Storage Sequence Control, located in each memory chassis, is a chain of FF's whose outputs drive slave inverters, generating a series of timing signals which direct the basic cycle of the storage modules. The Bank GO pulse starts the sequence by setting the Read FF and this pulse is transferred to successive FF's at 50 ns intervals. Each FF is set for 400 ns (Figure 3-7). The Bank Free condition is dropped as soon as the Read FF is set.

a) Read Drive. The clear output of the Read FF starts the 400 ns read drive. This is sent to the five storage modules and turns on read current in one core of each plane (H & V wires) already selected by the address. This enables data to be read out of memory.

b) Sense Drive. 200 ns after the read drive, a 100 n s sense signal is generated to sample the differential sense amplifier which receives the data word read from storage on double-ended sense lines. This sample pulse accomplishes two operations:

1. Samples the data read out of memory.
2. Allows a bank merge at the Data Exit/Entry Buffer Register (SE modules) common to the four banks and gates data to the data distributor and restoration register.

c) Inhibit Drive. The timing chain initiates the inhibit drive 475 ns after the read drive. During this 400 ns before the start of the inhibit drive, data has been read out of memory and transmitted through two holding registers common to all four banks. The first, H1 - H15 (EVEN/ODD CH.), holds the true value of the data word read out and the second G1-G19 the complement.

"Start Inhibit" gates the complement into the restoration register (PZ modules) which also stores the complement value. Inhibit current will or will not be turned on depending on the state of each bit position of the word to be stored ("0" or "1") respectively. To write a zero into memory the inhibit current controlled by the restoration register content (complement) is turned on and opposes the write current. For a 1, inhibit current is voided. 900 ns after the read drive an end inhibit signal clears the restoration register and ends the inhibit drive. Keep in mind that the PZ modules hold the complement. A "1" therefore enables inhibit, a "0" disables inhibit.

d) Write Drive. The timing chain transmits the write enable signal 500 ns after the read drive. This is sent to the five storage modules and like the read drive, turns on write current

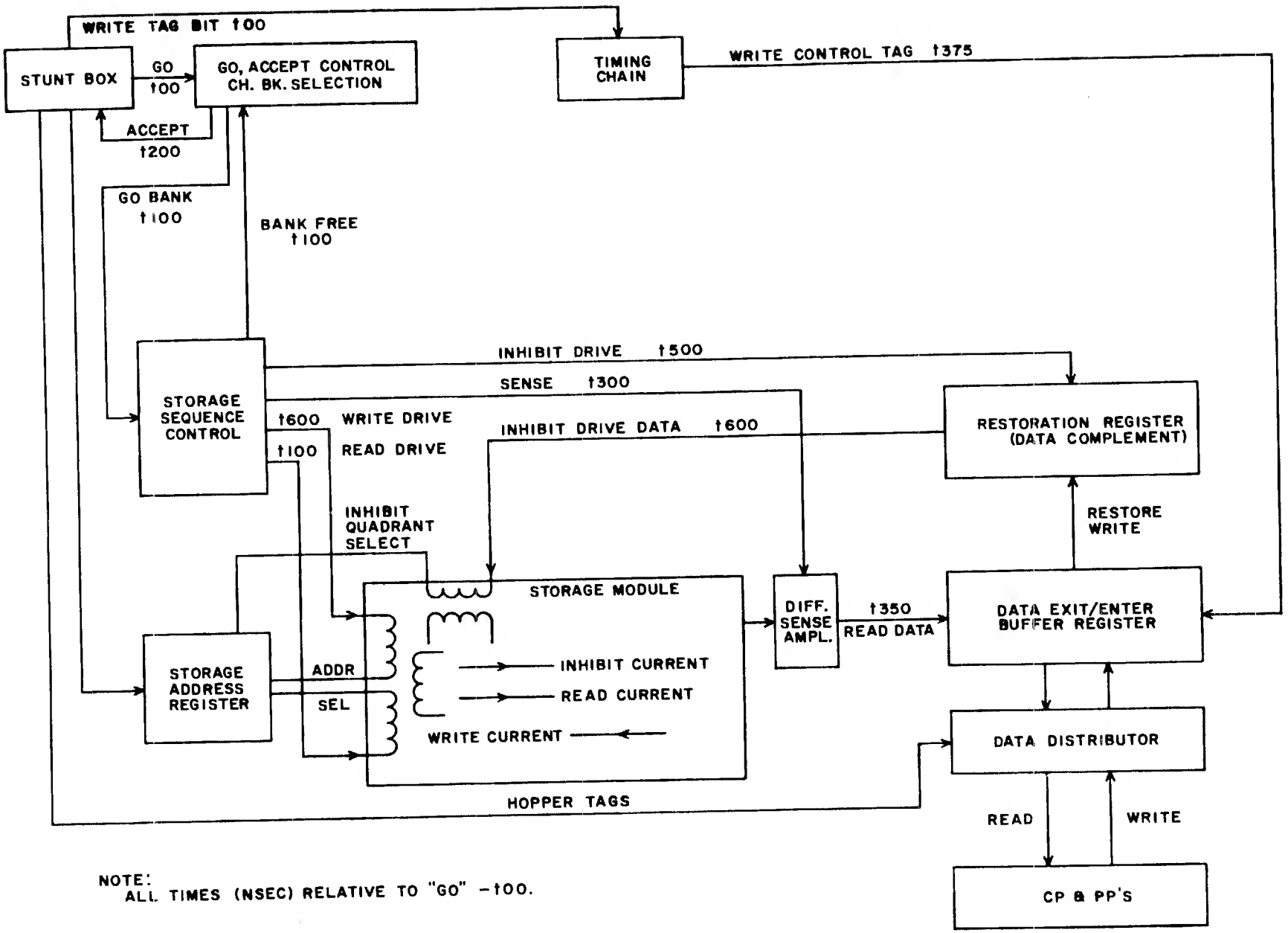


Figure 3-6. Memory Cycle Block Diagram

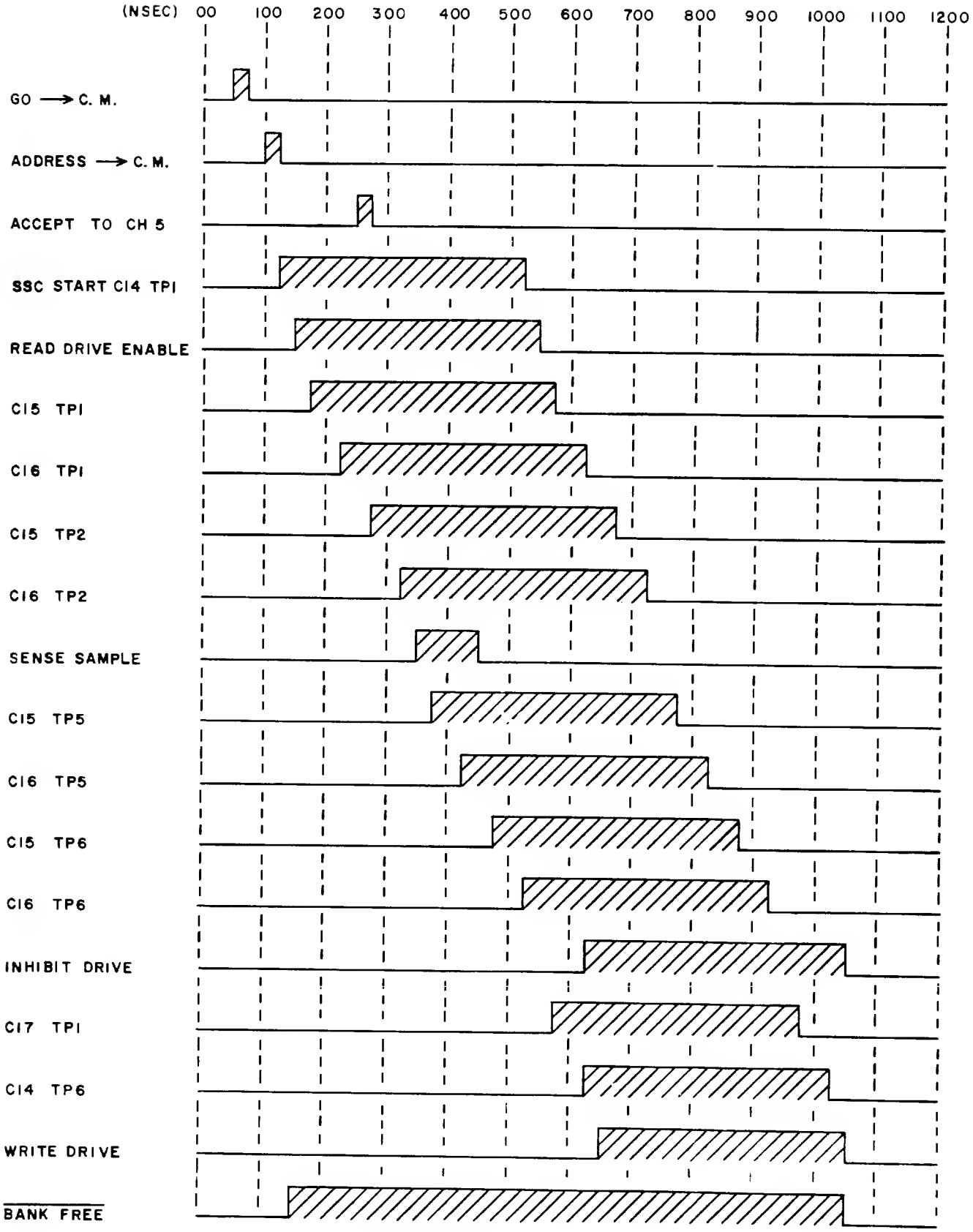

CENTRAL MEMORY BANK CONTROL (BANK 00)
(NSEC) 00 100 200 300 400 500 600 700 800 900 1000 1100 1200
GO → C. M.
ADDRESS → C. M.
ACCEPT TO CH 5
SSC START CI4 TPI
READ DRIVE ENABLE
CI5 TPI
CI6 TPI
CI5 TP2
CI6 TP2
SENSE SAMPLE
CI5 TP5
CI6 TP5
CI5 TP6
CI6 TP6
INHIBIT DRIVE
CI7 TPI
CI4 TP6
WRITE DRIVE
BANK FREE

Figure 3-7

in one core of each plane (H and V wires) already selected by the address. The write current flows in the opposite direction of the read, to switch the core to a 1. If a 0 is to be written, inhibit current opposes the write current and the core does not switch.

MEMORY READ

As outlined above, the following is the sequence of operation:

1. GO ⟶ Chassis and Bank Selection; Accept to Stunt Box.
2. Address ⟶ Storage. Selects particular core in each plane, i.e., 60 planes in 5 twelve-plane modules.
3. Read Drive. Reads data out of memory.
4. Sense. Samples data and transmits it to the data Distributor and restoration register.
5. Inhibit and Write Drive. Rewrite what has been read out.

MEMORY WRITE

The memory write is similar to the read up to Step 4. For every write operation a write signal (tag) is issued from the Stunt Box 150 nsecs. after the address. This passes through a timing chain and is so timed, that after Step 4 of the memory read has been accomplished, the write control signal clears the Data Exit/Entry Buffer register to allow the word to be written, to be gated in from the data distributor. The word that has been read out already is destroyed in the data distributor due to a lack of destination. Step 5 as outlined in Memory Read is now executed and the new word written into memory.

A word from the data distributor during a write reference goes to the Data Exit/Entry Buffer register on all chassis and then follows the restore path for writing in memory. Only one of the many banks is in the proper time spot in its storage cycle to store the word received, and this bank is the one associated with the write address.

To prevent writing into an address location before a previous read reference has been accomplished, the hardware in the Stunt Box never allows both a read and write to be in the Hopper at the same time. The exception is that if there is a PP central memory read in the hopper, a CM. write from the Central Processor can gain priority into the hopper. Software should prevent a CM write in a location that has not yet been read by a previous PP instruction.

BANK FREE SIGNAL

The bank free condition is established when all FF's in the timing chain are cleared, i.e., no pulse travelling down the chain. The Read FF (first FF in chain), an intermediate FF and the Write FF (last in chain) contribute timing signals to the bank free circuitry (G9,

G19/G29, G26 EVEN/ODD CHASSIS) and indicate whether or not a pulse is in the chain. All these FF's must be cleared to signal bank free at the end of the storage cycle.

DATA DISTRIBUTOR

The data distributor essentially acts as a post office, distributing data to and from CM, the maximum transfer rate being 100 ns. The distributor routes data to and from proper origins and destinations as directed by control information or tags entered in the stunt box along with each address. The tags serve to identify the address sender, origin or destination of data, and nature of the address, e.g., read, write, or PP exchange jump.

CONTROL TAGS

The six-bit tag is sent from the Stunt Box to the tag translator. After translation the tag is sent to the data distributor to identify the origin and destination of data. For any write reference a write tag bit is sent directly from M1 in the stunt box through a timing chain to the Data Exit/Entry Buffer register, enabling the data from the write distributor into the restoration register.

READ DISTRIBUTOR

The read distributor accepts read words from the 8 CM chassis and routes them to the several destinations.

The distributor is organized on chassis 3, 4, 9, and 10, each of which handles 15 bits of the 60-bit word. Chassis cable limitations dictate the organization. The listing below shows the bits handled by each chassis.

CHASSIS	BITS
3	0-14
4	15-29
9	30-44
10	45-59

Thus, chassis 13-16 (Figure 3-8) each send the same 15-bit group to chassis 3, 4, 9, and 10. A read word from chassis 3 retains bits 0-14, but sends remaining bits in three groups to chassis 4, 9, and

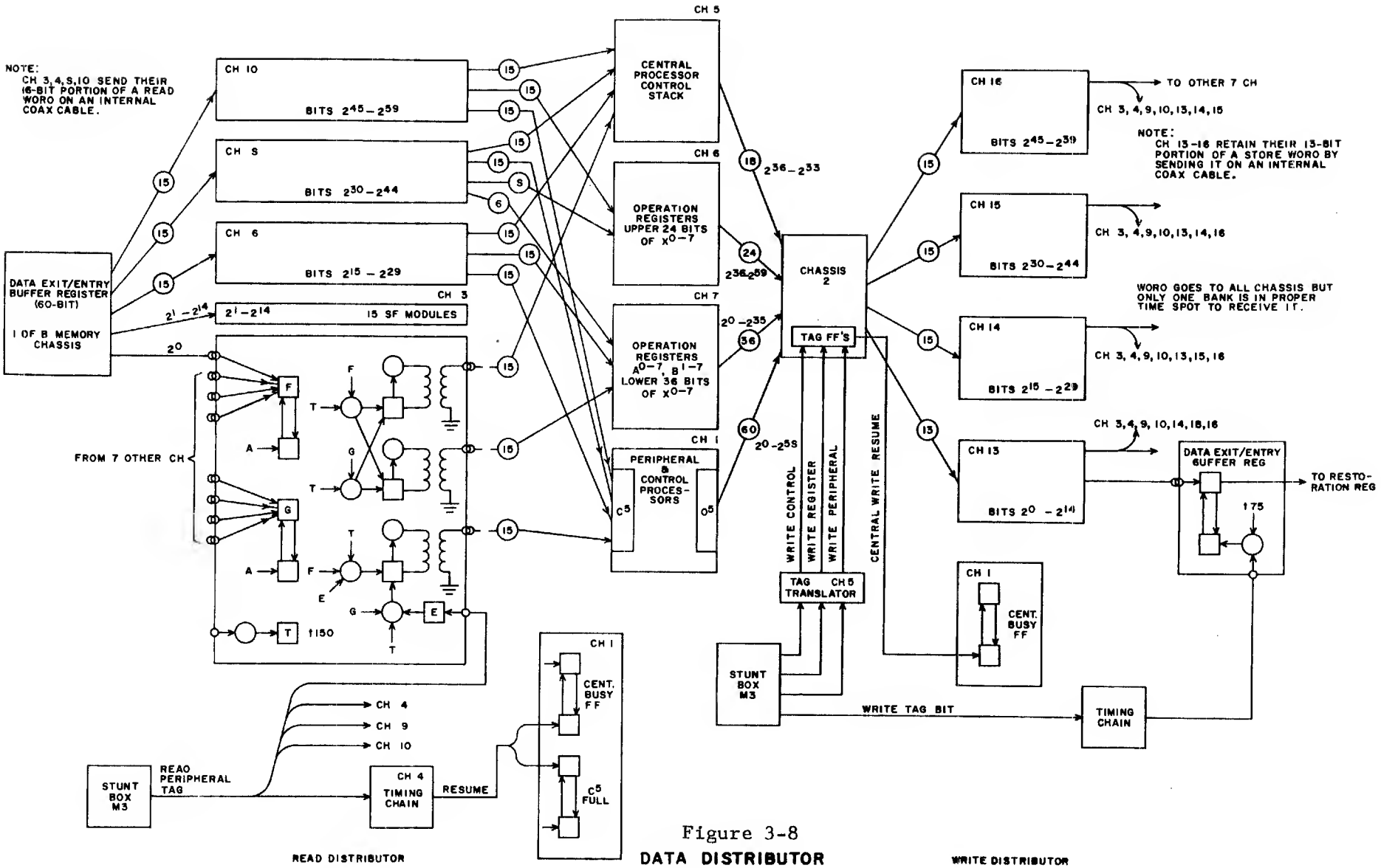


Figure 3-8
DATA DISTRIBUTOR

10. The tag gates the read word to the C5 register in the read pyramid on PP chassis 1.

The read peripheral tag also enters a time delay chain and is returned to the PP as a resume signal. The resume sets the C5 full FF in the PP (after data word is in C5) to signal the presence of the read word. The same resume also clears the Central Busy FF to indicate to PP control that the address has been accepted by the stunt box and CM has delivered the word. This allows the Peripheral Processors to send another address to the stunt box.

WRITE DISTRIBUTOR

The write distributor accepts words from the several sources and stores them in 1 of 8 memory chassis. The distributor is on chassis 2. The 60-bit word on chassis 2 is split into four 15-bit groups which are sent to chassis 13-16 respectively. Each of these chassis in turn sends (or stores) its 15-bit group to the other 7 chassis (Figure 3-8) unconditionally. The JH modules (I28-I37) on chassis 13, 14, 15, 16 have 12 outputs, 8 for one bit sent to all 8 CM chassis and 4 for another bit sends to 4 chassis only. The other four chassis are fed with this same bit from another JH module.

Figure 3-8 shows a detail of the write path. The 3-to-1 fan in on chassis 2 selects the proper word under control of the store tag from the stunt box which is established ahead of the data. The word is then split and transmitted to chassis 13-16. The chassis 2 data registers and the tag FFs are cleared simultaneously.

One minor cycle after the register clear, a central write resume is sent to the PP to clear the central busy FF. This allows the PPs to send another address to the stunt box.

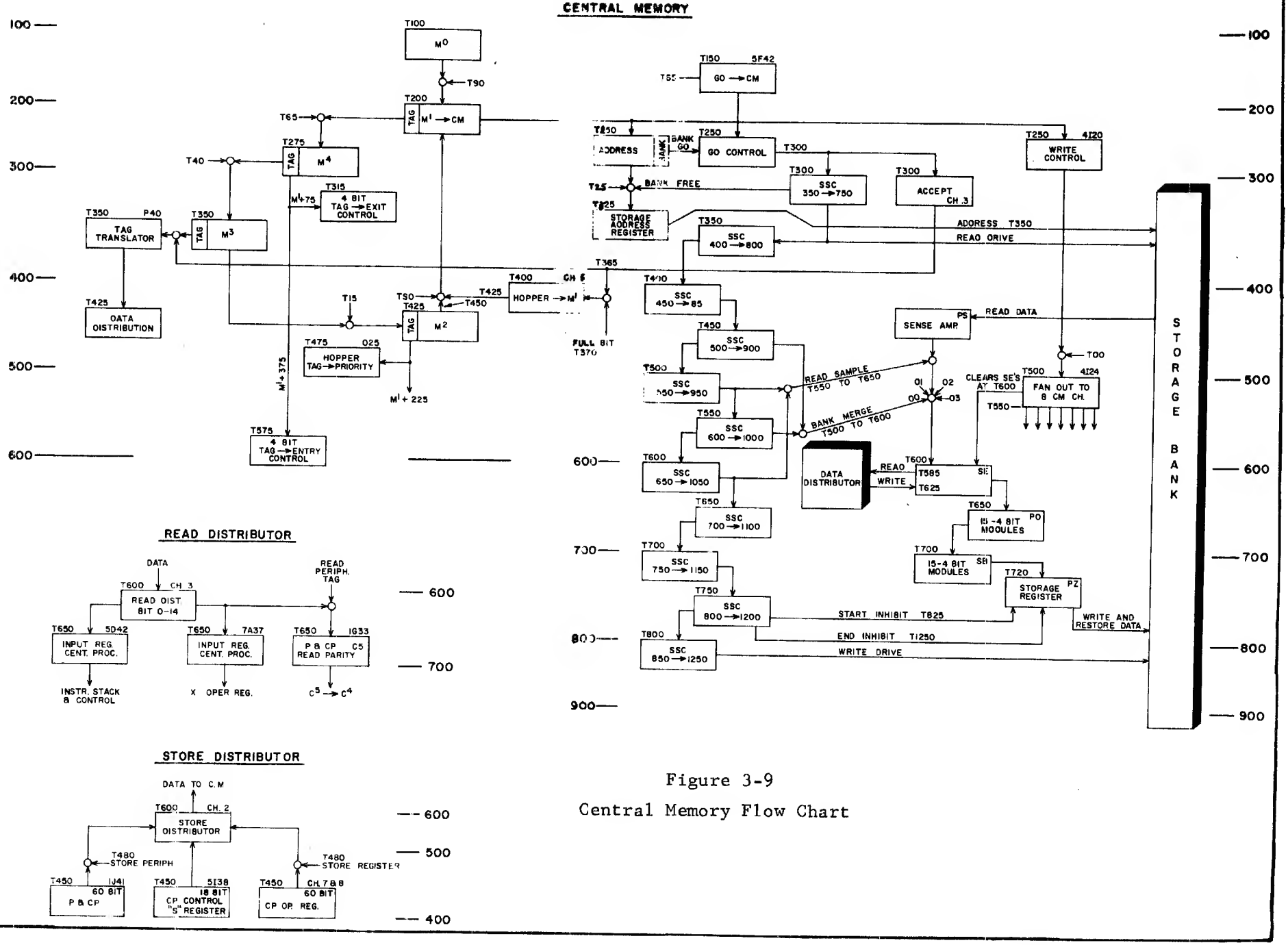
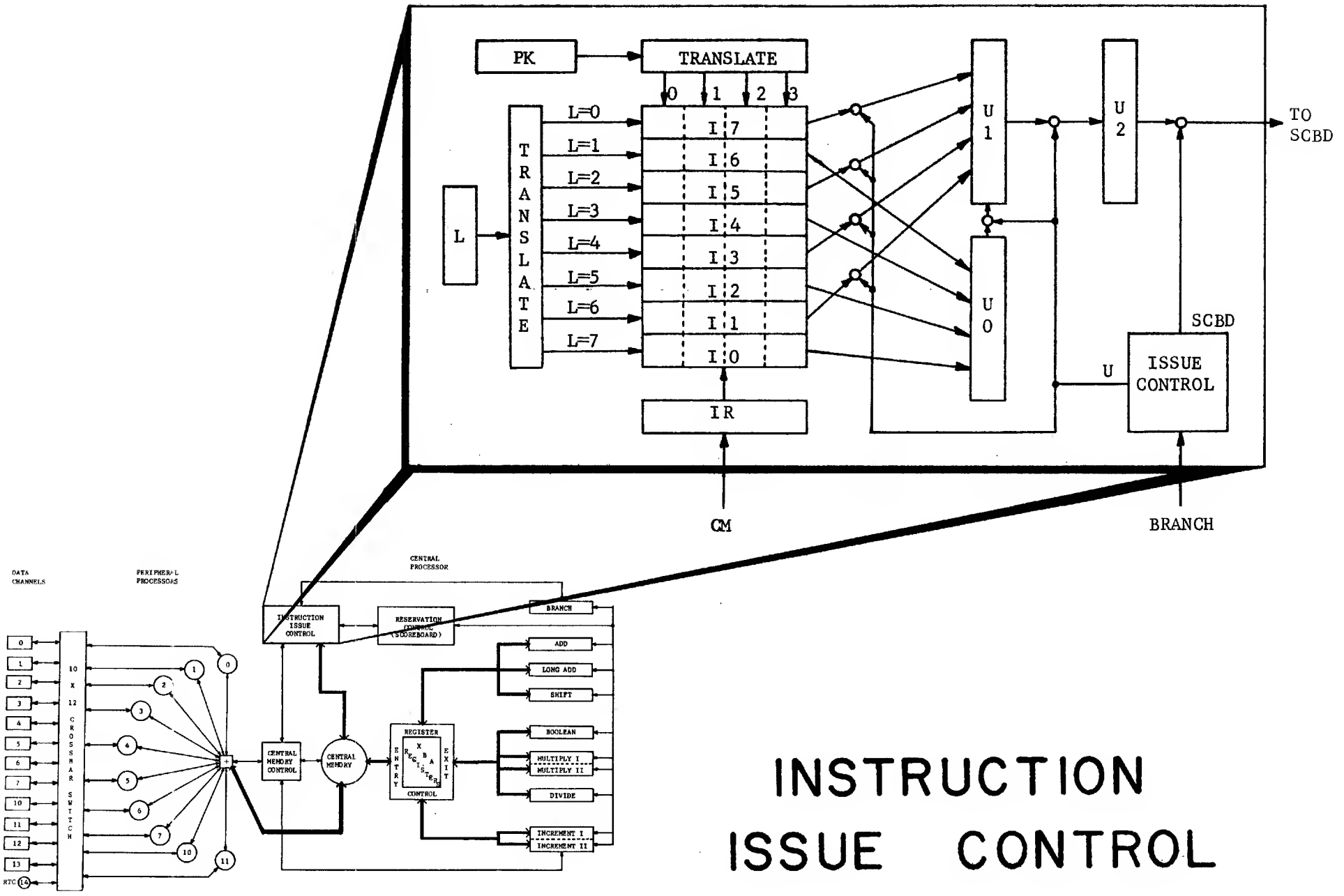


Figure 3-9

Central Memory Flow Chart

CHAPTER IV

INSTRUCTION ISSUE CONTROL



INSTRUCTION ISSUE CONTROL

CHAPTER IV

INSTRUCTION ISSUE CONTROL

INTRODUCTION

The 6600 central processor, designed for every high speed computing, uses parallel functional units and overlapping memory references to reduce program running time. Another method of reducing time consumed during execution of a program eliminates unnecessary references. The following example of a portion of a typical scientific program illustrates the desirability of minimizing memory references:

EXAMPLE:

ADDRESS	
100	Load operand A modified by B1
101	Multiply by operand C
102	Subtract operand D
103	Jump if sign of results are negative
104	Store results in X modified by B1
105	Update B1
106	Jump to address 100

The above loop example shows that memory references to obtain the same group of instructions are necessary for each pass through the loop. To avoid unnecessary memory waiting time, the central processor uses an instruction stack to hold short loops.

INSTRUCTION STACK

DESCRIPTION

The central processor instruction stack is made up of eight 60-bit instruction registers, labeled I0, I1, I2 . . . . . I7. An additional 60-bit register known as the Input Register serves as an input buffer between central memory and the instruction stack.

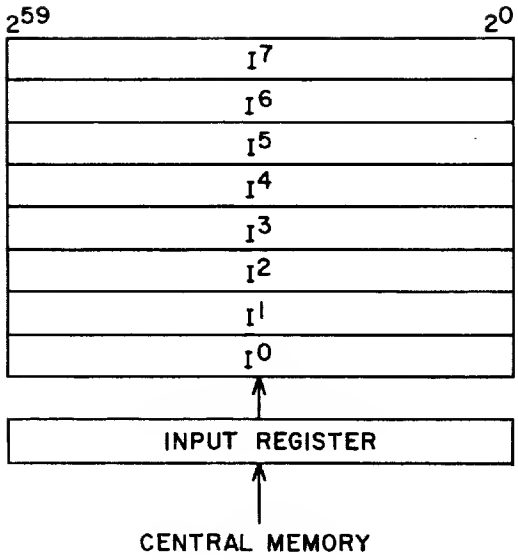


Figure 4-1. Instruction Stack

Instruction formats for any 60-bit instruction word in the stack may be 15-bit and 30-bit instructions in any of the combinations diagrammed in Figure 4-2.

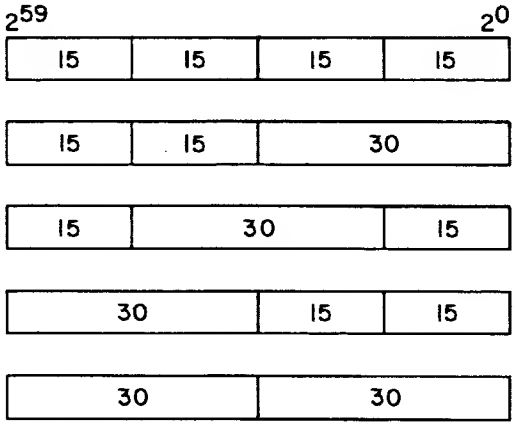


Figure 4-2. Possible Instruction Word Formats

The maximum capacity of the instruction stack is, then, thirty-two 15-bit instructions. A program loop that can be contained within the stack can be executed as many times as necessary without making any memory references for instruction words. Note that an iterative loop cannot be 32 instructions and remain in the stack since: a.) a jump instruction comprising part of the loop is a 30-bit instruction, and b) a jump cannot be executed from I0 (the bottom register of the stack.)

OPERATION

The instruction stack is loaded one instruction word at a time as the instruction words are read from central memory to be executed. Each instruction is loaded into I0, sent to the translating networks, and then moved to I1. As each instruction word enters I0 and is moved upward in the stack, the complete stack contents must move upward to make room for the new instruction word. This upward movement of stack contents is called inching. As the stack contents are inched, the top instruction word (in I7) is discarded.

NOTE

Throughout the descriptive material which follows, references are made to specific FFs, logical networks, etc. To aid in understanding the text, refer to text diagrams and to 6600 Central Processor diagrams.

INCHING

The process of inching the instructions in the stack is controlled by the inch counter. This counter is a 2 rank, 2 bit, binary counter initiated by the following conditions:

a) next instruction word is in I0, and
b) first instruction from I0 is being issued to the translating networks (U registers).

Once the inch counter is started, it continues counting until the entire stack has been moved up. During the inching process, the instruction word in I7 is discarded.

PARCELING

Since there are 15 and 30-bit instructions which can be combined in an instruction word in any order, the problem of extracting instructions from the ranks of the stack is encountered. The scheme used to extract instructions is called parceling and is controlled by a 2 rank, 2 bit,

binary counter called the parcel counter (PK). A 60-bit instruction word is partitioned into 4 parcels, designated 0, 1, 2, and 3. (Refer to Figure 4-3). Each parcel is 30 bits in size to ensure that a

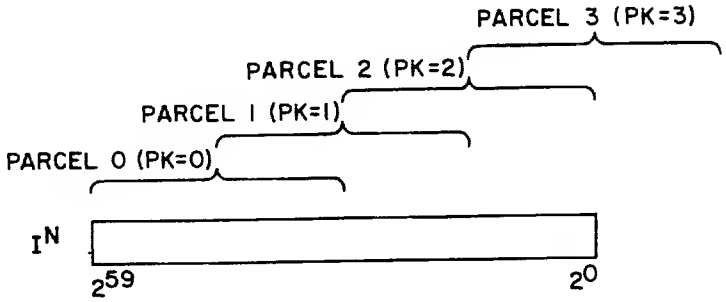


Figure 4-3. Instruction Word Parcels

complete instruction is obtained when extracting a parcel from the word. In many cases (e.g., a 15-bit instruction in a parcel), unusable information is extracted along with the meaningful. In these cases, unusable information is discarded as the instruction is placed in the U registers.

LOCATOR

In most cases, instructions that are sent to the U registers come from the lowest rank of the stack (I0). However, an instruction word inched up into the stack can still be issued to the U registers for execution. Thus, if inching occurs before an entire instruction word is issued (parceled to the U registers, or if an iterative loop in the stack is being executed, the next instructions will come from some stack register other than I0. A Locator network (L) keeps track of the register in the stack holding unissued instructions (i.e., the register from which the next instruction will come).

ISSUE CONTROL

The movement of instructions from the stack and through the U register is controlled by a circuit called Issue Control. There are two basic signals associated with Issue Control:

a) U issue (i.e., U1 $\longrightarrow$ U2 and I^n $\longrightarrow$ U1)
b) scoreboard issue (i.e., U2 $\longrightarrow$ Scoreboard)

The Issue Control circuit controls the operations of the parcel and inch counters, the L network, instruction issue and movement from the instruction stack to the scoreboard. The availability of functional units and operating registers, and information on instruction translations is monitored by Issue Control to assist this circuit in its functions.

INSTRUCTION REGISTERS

From the instruction stack, instructions flow in sequence to two registers to be translated. (Refer to Figure 4-4).

The first register, U1, consists of 30 FFs. Entry into U1 is controlled by the L counter and by the parcel counter. Instructions which come from even stack registers are transferred via an auxiliary register, U0. (Note that this does not affect the transfer time from the stack to U1; instructions from even and odd ranks of the stack arrive in U1 at the same time (t80)).

The fm portion is now translated to set the proper Select, Request and Result FFs. (Refer to Figure 4-5)

At approximately the same time these FFs are set, the transfer to U2 is performed (t15-25). There are 3 different cases:

a) 0X - Instructions which contain the branch address in the K portion.
b) 30-bit instructions which contain one of the two operands in the K portion.
c) 15-bit instructions which do not use K.

(Refer to Figure 4-6)

The second register, U2, consists of 53 FFs, which hold:

a) Information about the Functional Unit request, and result and operand register selections gained from the translation in U1.
b) The instruction designator ($\overline{f}$, $\overline{m}$, i, j, k).
c) K portion.

This information is now translated and transmitted into the scoreboard (refer to Figures 4-4 and 4-7).

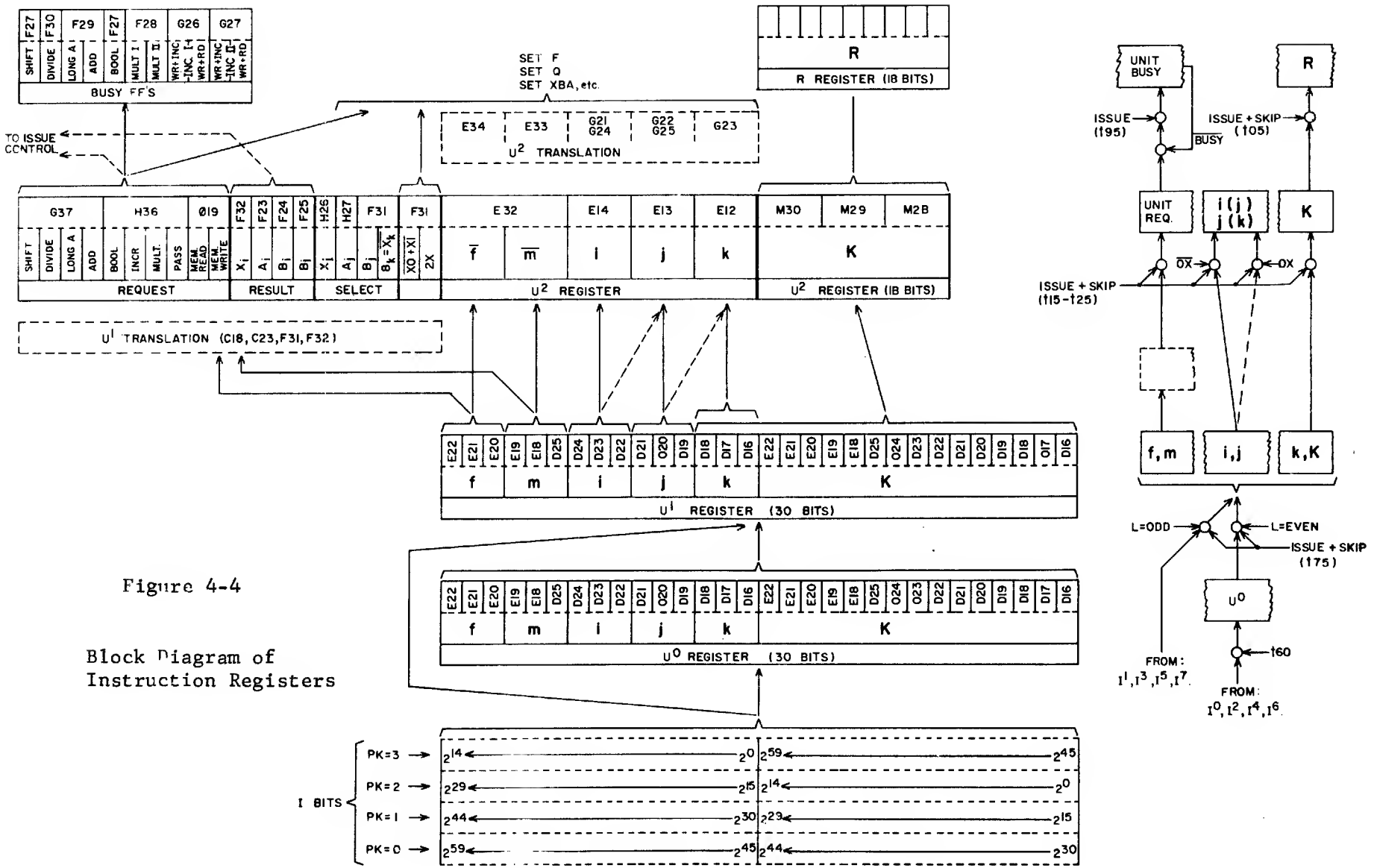


Figure 4-4

Block Diagram of Instruction Registers

U_2 FFS ↓ \ U_1 fm →		0___ 0 1 2 3 4 5 6 7	1___ 0 1 2 3 4 5 6 7	2___ 0 1 2 3 4 5 6 7	3___ 0 1 2 3 4 5 6 7	4___ 0 1 2 3 4 5 6 7	5___ 0 1 2 3 4 5 6 7	6___ 0 1 2 3 4 5 6 7	7___ 0 1 2 3 4 5 6 7	
UNIT REQUEST	SHIFT			XXXXXXXX		X				G27
	DIVIDE					XX X				
	LONG ADD	X			XX					
	ADD				XXXXXX					
	BOOLEAN		XXXXXXXX							H36
	INCREMENT	X XXXX					X←IF i=0→X	XXXXXXX	XXXXXXXX	
	MULTIPLY					XXX				
	PASS					X				
REQU.	MEM. READ						X←IF i=1_5→X			Ø19
	MEM. WRITE						X←IF i=6+7→X			
RESULT	X_i		XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXX X			XXXXXXXX	F32
	A_i						XXXXXXXX			F23
	B_i							XXXXXXXX		F24
	B_j			XXX						F25
SELECT	X_j	X X	XXXXXXXX		XXXXXXXX	XXX XXXX	XX	XX	XX	H26
	A_j						X XX	X XX	X XX	H27
	B_j	X XXXX		XXXXXXXX		X	X XX	X XX	X XX	F31
	$B_k = \overline{X_k}$	X XXXX					XXXXXXXX	XXXXXXXX	XXXXXXXX	
	$\overline{XO + XI}$	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXX	XXXXXX	XXXXXX	XXXXXX	F31
	2X			XXXXXXXX						

Figure 4-5. U1 Translation

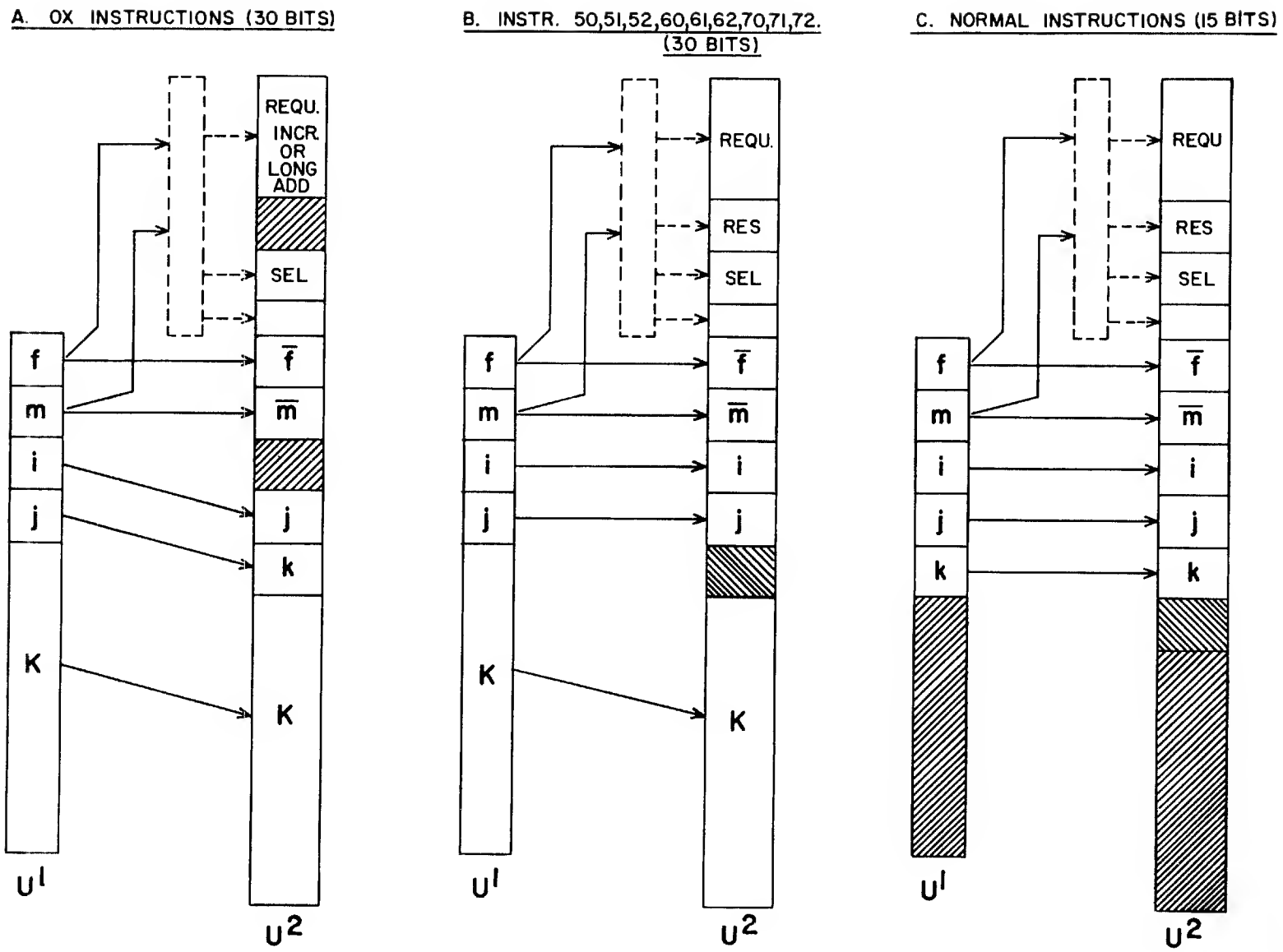


Figure 4-6. U1 ⟶ U2 Transfers

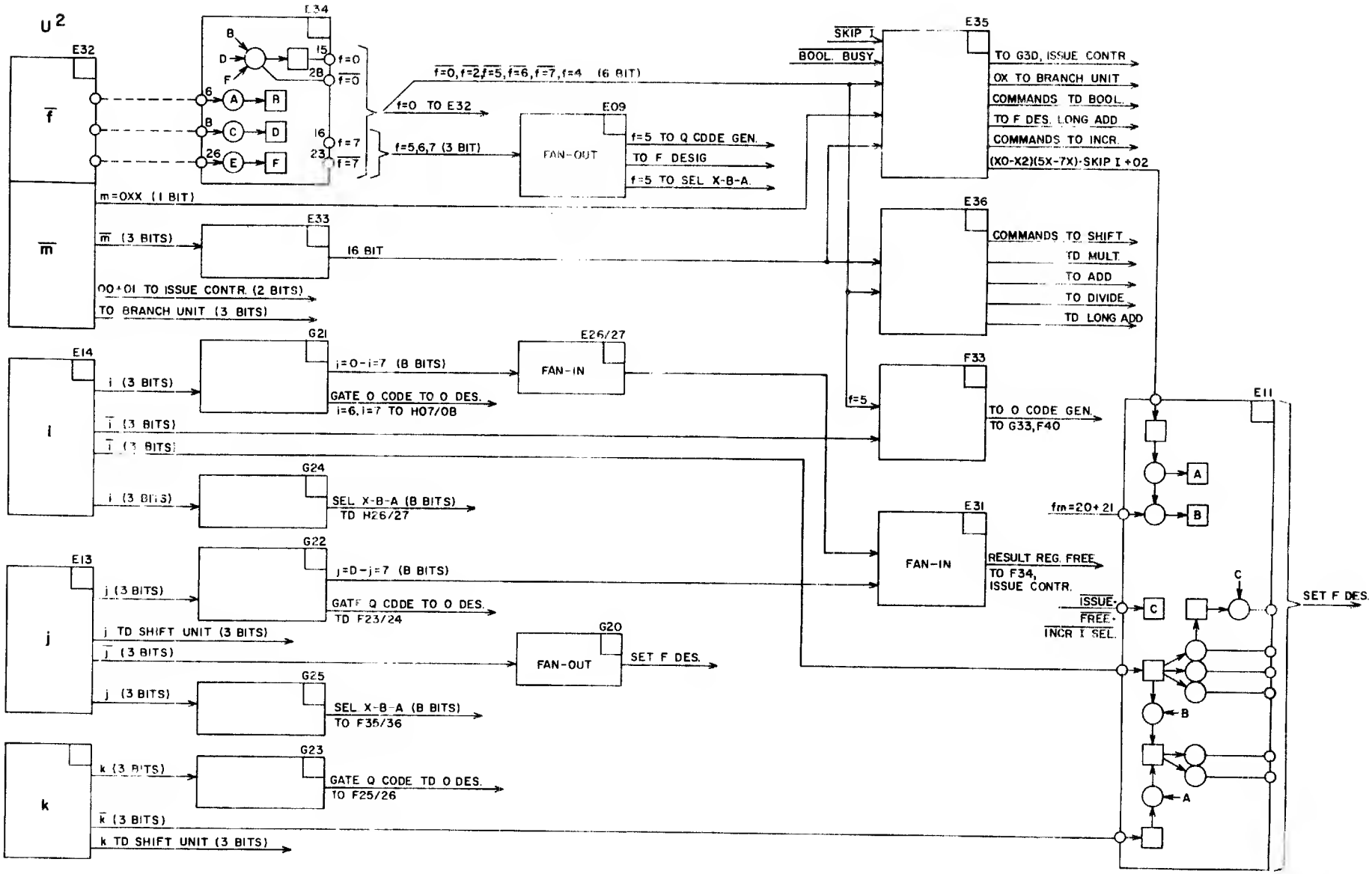


Figure 4-7. U2 Translation

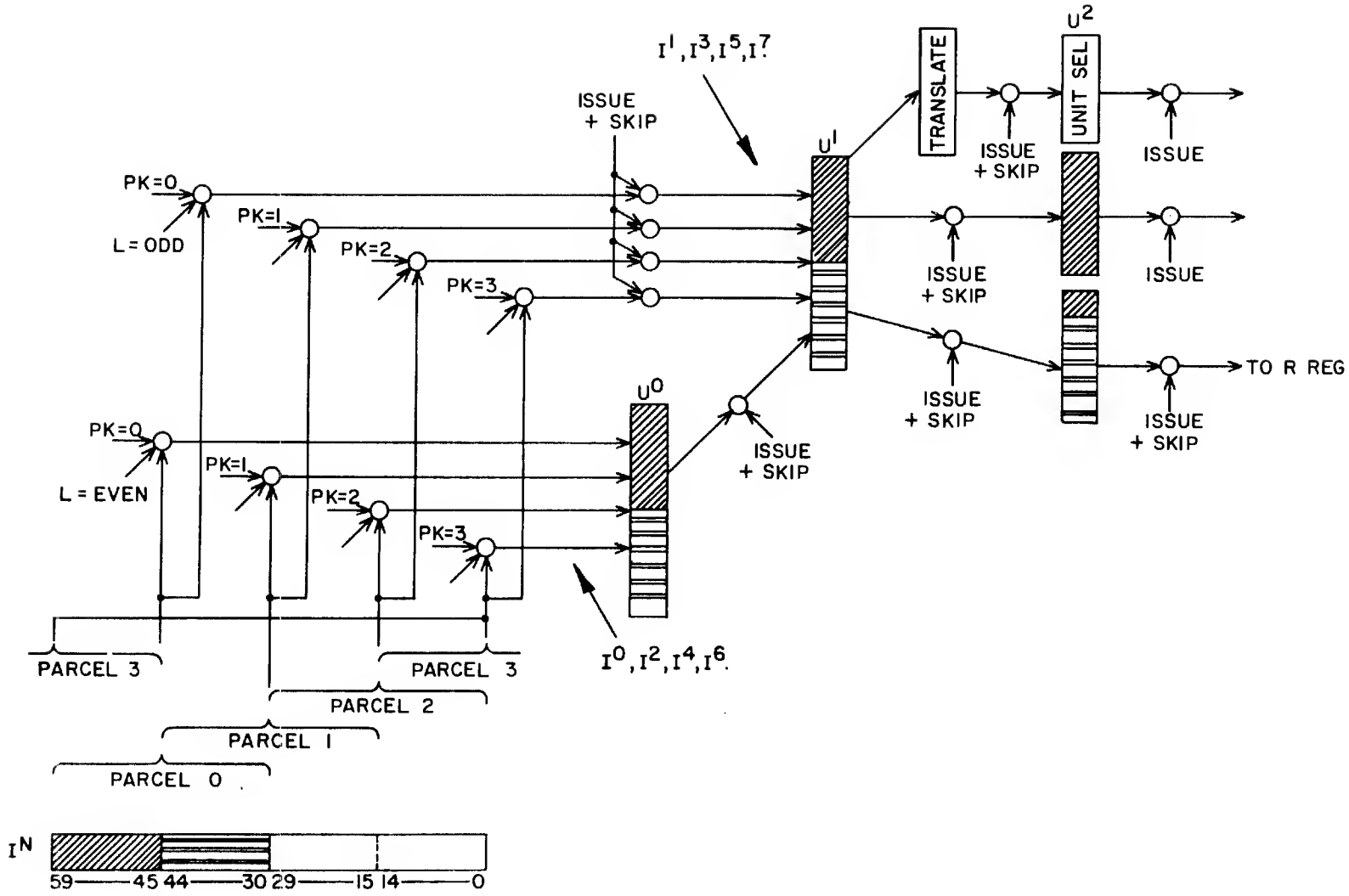


Figure 4-8. Parcel Counter

PARCEL COUNTER

The parcel counter is used to extract 4 parcels of 30 bits each from I^n. The 4 parcels are needed in order to pick out any 30-bit or 15-bit instruction combination which could be contained in an I register and send these instructions to the Scoreboard.

15-BIT INSTRUCTIONS

With the first issue pulse, parcel 0 is sent from I^n to U1 and the parcel counter is advanced to 1. The second Issue Pulse transfer U1 to U2 and parcel 1 to U0. The ensuing Scoreboard Issue sends the contents of U2 to the Scoreboard, parcel 1 to U2 and parcel 2 to U1. (Refer to Figures 4-9 and 4-10)

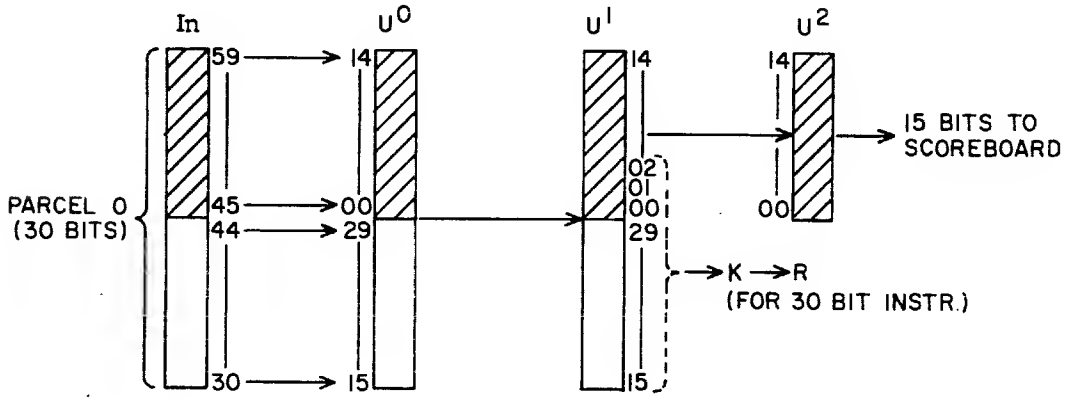
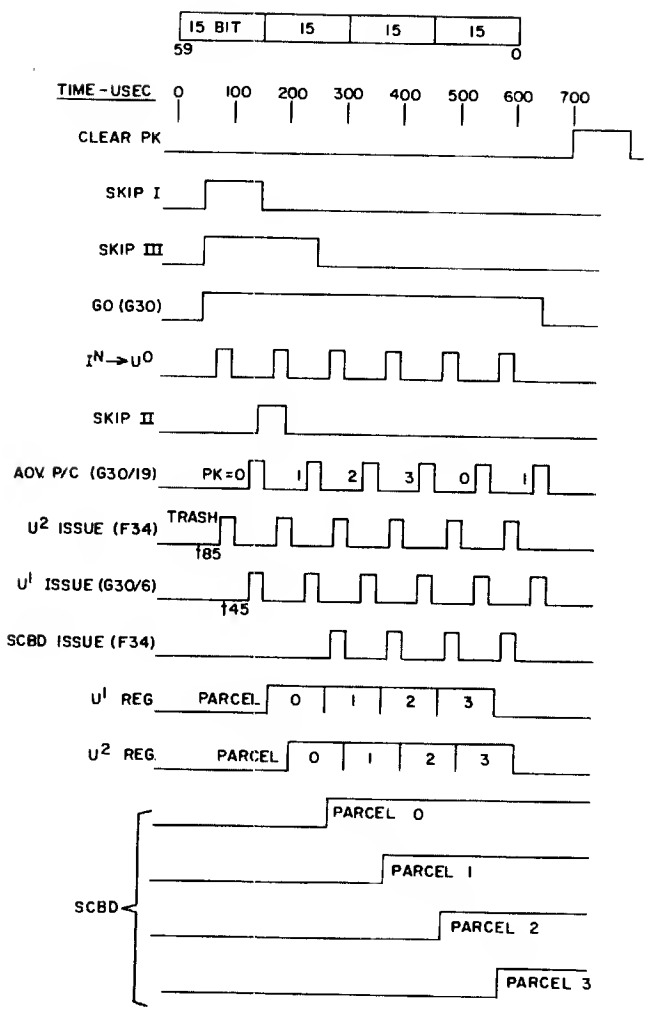


Figure 4-9. Parcel Path to Scoreboard

After all parcels have been issued to the Scoreboard, the parcel counter will be at count 1. This enables a "0" into F37/P3. Because FFs TP/5 and TP/6 have been set after inching of I4 to I5, a "0" from F37/P25 goes through H25 and G36 and sets the Stop FF on G30, preventing any more Scoreboard issue pulses. Also, FF A on F37 will be set giving a "1" out of F37/P23, setting the FF TP6 on R32 at t50. One output of this FF is used to send a clear pulse to the Parcel Counter on G31, setting the counter back to 0.

30-BIT INSTRUCTIONS (50-52, 60-62, 70-72)

The proceed signal coming into G30/P11 sets Skip I, III and the GO FF, resulting in a U2 Issue signal from F34. This enables an Advance PK pulse from G20/P19, and sets the PK to 1. The same Issue pulse also enables parcel 0 into U1. The next Issue pulse (100 ns later)



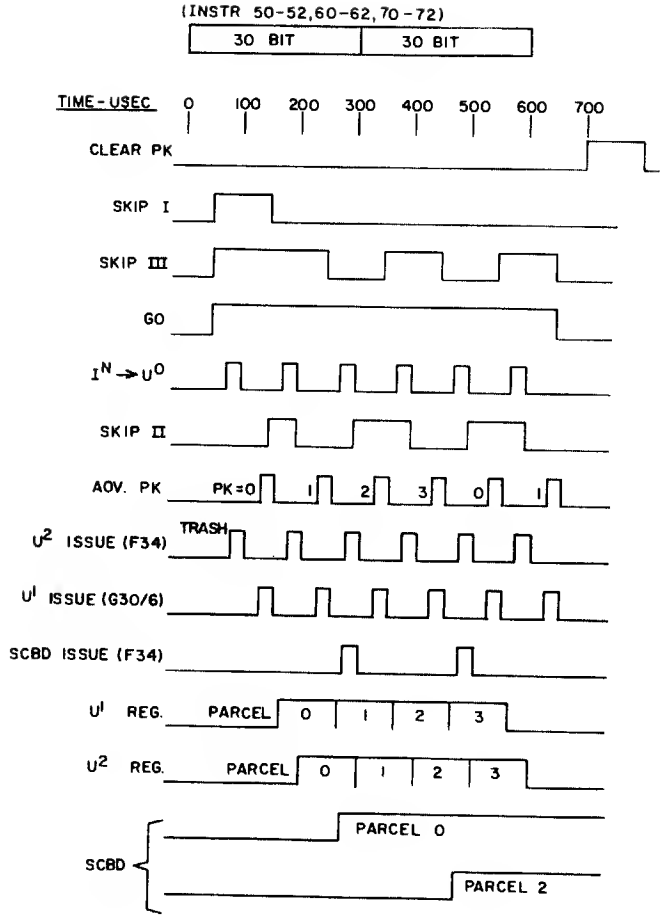


Figure 4-10. Instruction Issue Timing

transfers U1 ⟶ U2, parcel 1 into U1 and advances PK to 2. By this time, the Skip III FF will have been cleared, allowing a Scoreboard Issue pulse.

This pulse transfers the 30-bit parcel 0 from U2 to the Scoreboard. It also allows the Skip II FF to reset. The 3rd Issue pulse sets PK to 3, transferring parcel 1 to U2, parcel 2 to U1. By now, the Skip II FF is set, disabling any Scoreboard Issues pulses. The 4th Issue pulse transfers parcel 2 to U2 thereby destroying parcel 1, which was still in U2. The only difference between a 30-bit and 15-bit instruction is that the Skip II FF is set for 1 minor cycle to prevent the second Scoreboard Issue pulse and therefore does not send parcel 1 to the Scoreboard.

30-BIT INSTRUCTIONS (0X)

The first Issue pulse will again transfer U0 to U1 and advance PK to 1. The second Issue will be fed into G30/P23 together with the 0X translation from U1. This clears FF/E, disabling a second Advance PK pulse and also disabling the P and L incrementers. This Issue also transfers U1 to U2 and parcel 1 into U1. Together with the third Issue, a Scoreboard Issue transfers U2 to the Scoreboard. This Scoreboard pulse, together with the 0X translation from U2, is fed into H05/P9 setting the Stop FF. This disables any more Scoreboard Issue pulses. Instruction issue control now waits for an indication from the Scoreboard as to whether the jump condition was met or not. (Refer to Figures 4-11 and 4-12).

INCH COUNTER

The inch counter is used to shift 60-bit instruction words upward in the instruction stack (I0-I7). When the parcel counter = 0, L counter = 7 and t = 60, and an Issue or Skip signal occurs, a "0" into G29/11 sets the Inch FF. This FF sends an advance pulse to the inch counter. Before the counter is advanced to 1, gates C and D on G29 are enabled setting FF/TP1. One output of this FF will set the Program Address FF on P37, starting a new RNI. Another output first clears the I7 register and then enables the transfer of I6 ⟶ I7. Another pulse from this FF, at a slightly later time, clears the I6 register and transfers I5 ⟶ I6. (Refer to Figure 4-13)

By now, the inch counter is set to 1 enabling the transfer of I4 ⟶ I5 and I3 ⟶ I4 etc. After the last transfer has been enabled (I0 ⟶ I1), the counter is set at 3. The 4th advance pulse now sets the counter back to 0 and the Inch FF is cleared.

The counter is again ready to begin a new transfer sequence. The instruction word now in I7 is discarded when the next I6 ⟶ I7 transfer occurs.

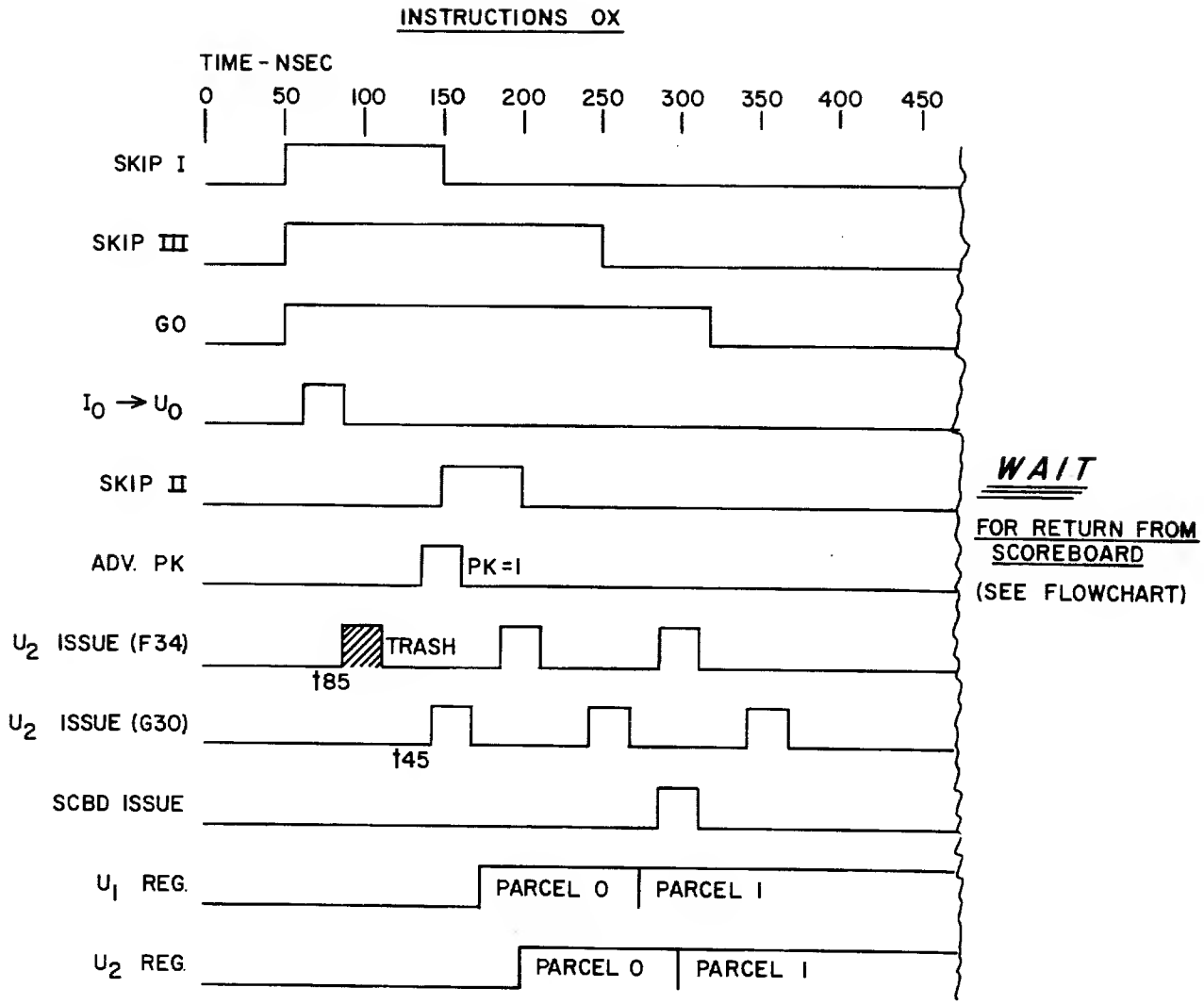


Figure 4-11 OX Instruction Issue Timing

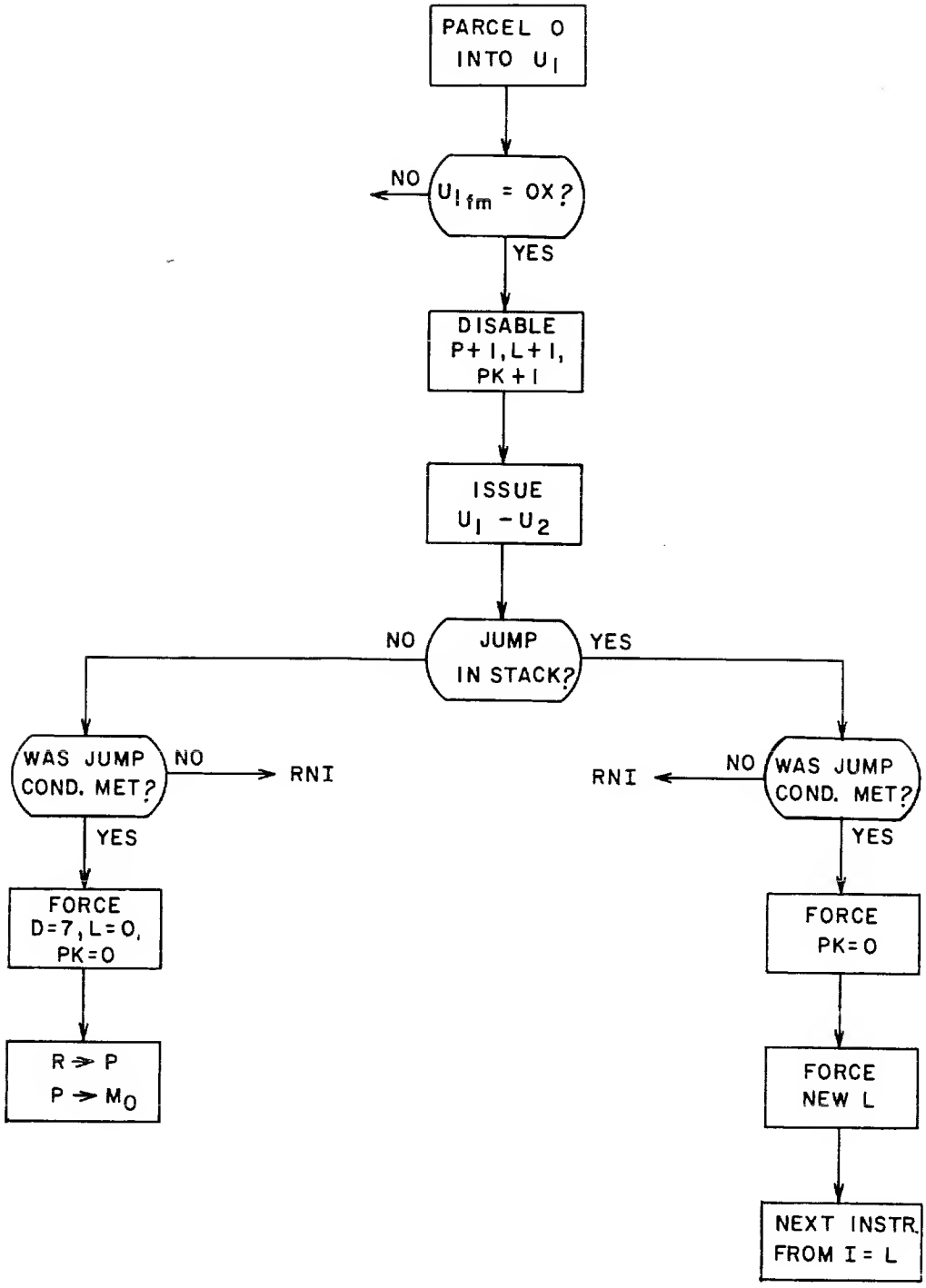


Figure 4-12. 03-07 Instructions

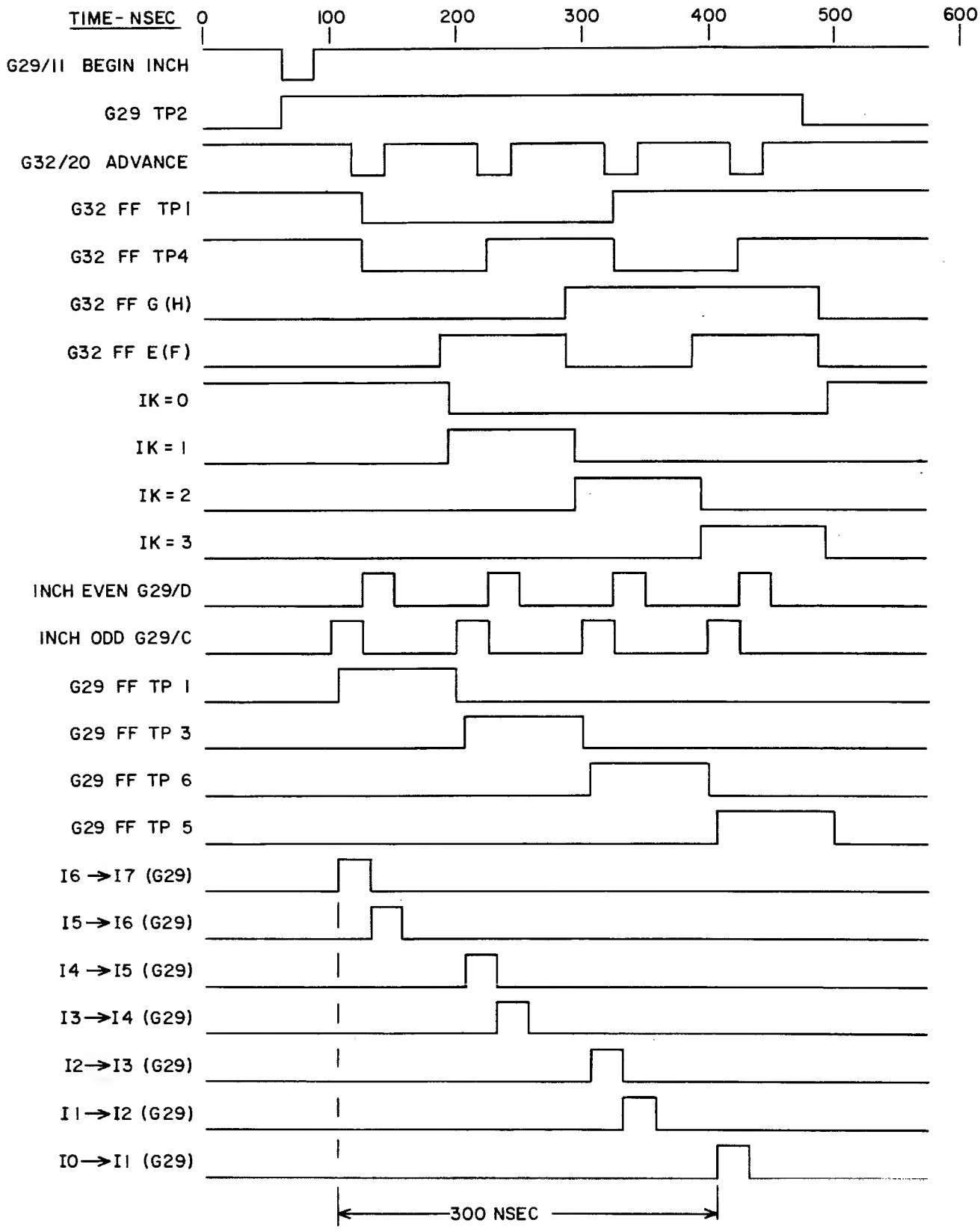


Figure 4-13. Inch Counter

LOCATOR COUNTER

To avoid confusion in the L counter discussion which follows, the following statements should be noted:

1) The L counter FFs on H28/TP1,2,5 contain the complement of the L count. (e.g., all set: L count = 0; all clear: L count = 7.) Note the distinction between L <u>counter</u> and L <u>count</u>.
2) Reduce L means the L <u>counter</u> goes from 6 $\longrightarrow$ 5 or 3 $\longrightarrow$ 2.

The major function of the L counter is to guide the parcel extraction from the proper register (e.g., I0, I1 etc.) in the instruction stack; when L = 7, I0 will be selected, when L = 6, I1, etc. Assume L = 7 and PK = 0; this means that the upper 30 bits of I0 would be gated to U0. When PK advances to 1, the next parcel will be transferred etc. By now, the Inch counter also started the inching of I6 $\longrightarrow$ I7, I5 $\longrightarrow$ I6 etc. During this time L was equal to 7.

When the PK reaches 3, and no 0X (Branch) instruction is in the U1 translator, a "0" into G28/P17 will enable the H gates and disable the G gates.

The function of these H and G gates are to switch the L counter FFs from one loop to another. One loop called the L-T loop goes from H28 through H29, G36, G28 and back to H28. The second loop, called the Reduce L loop, goes from H28 through H30, G28 and back to H28. In a no branch instruction, the T FFs on H28 are all set, which means that in the L-T loop a zero is actually being subtracted from L. This permits holding L to its same value through the L-T loop as long as the G gates are made. Switching to the Reduce L loop also causes a "1" into G28/P12 (from G29/P18) disabling the H and G gates. This prevents FF L0 from holding its value and it is cleared at the next t50.

This means that the L count has advanced to 1. However, as soon as the GO FF on G30 is cleared, a "0" out of A on G36 will set all L FFs on H28 forming an L count of 0. Parceling of the next instruction word from I0 is now ready to begin.

In case of a conflict (instruction not accepted by the Scoreboard), all Issue pulses are stopped preventing Advance P and Advance PK. However, the Inching process will continue until completion. After the inching of I1 $\longrightarrow$ I2 is completed, the L counter is again advanced setting the count to 1; the count remains set at 1 since the GO FF remains set. This will allow a new instruction to be entered into I0 while waiting for the instruction to be accepted. Once the instruction is accepted by the Scoreboard, the following Issue will advance the Parcel counter again but the remaining parcels are taken from I1. It is now necessary to wait until all instructions out of I1 are accepted before starting a new Inching process and therefore a new RNI.

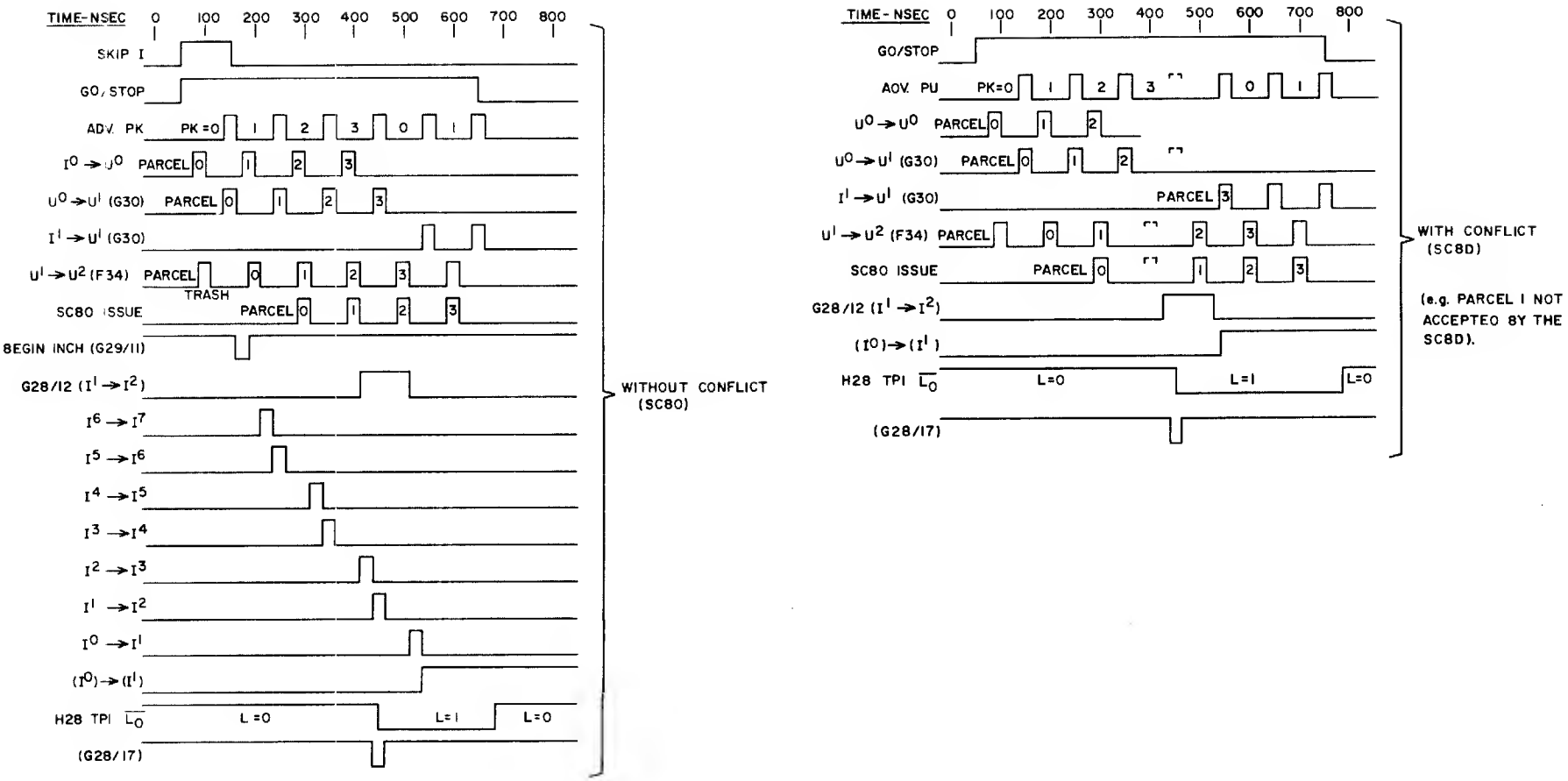


Figure 4-14. L-Counter

For branch instructions (if conditions are met and jump is in stack), the L-T loop supplies the new L count and sets the L FFs to the new value. After all the instructions in this new instruction stack register (defined by the L count) have been issued, a count enable, K on H30, together with the H gate on G28 allows the L count to be reduced by one, and so on until the count L is again equal to 0 $(\overline{L = 7})$.

At the beginning of the branch, the inching also transferred I0 $\longrightarrow$ I1 but now it is not possible to start a new inching process because L must equal 7. There is also only one RNI immediately after the inching has started but since inching cannot start a new RNI is not possible.

Remember:

To start Inching: $(PK = 0)(L = 7)$(Issue pulse)
To start RNI: Inch I6 - I7

ISSUE CONTROL

The Issue Control block diagram Figure 4-15 depicts the functions performed by the Issue pulses from F34 and G30. As not all of these pulses are produced by the same conditions, they are given the following names:

1) Pulses from F34/P9, 11, 13 are called: U2 Issues
2) Pulses from G30/P6 are called: U1 Issues

} also called: Issue+Skip

3) Pulses from F34/P8, 14, 19/26, 28 are called: Scoreboard Issues
also called: Issues

These pulses are produced if the following conditions exist:

U1 and U2 Issues = (Unit Request)$(\overline{\text{Unit Busy}})$(Dest. Reg. free)(GO) + Skip
Scoreboard Issues = (Unit Request)$(\overline{\text{Unit Busy}})$ (Dest. Reg. free)(GO)$(\overline{\text{Skip}})$

STOP INSTRUCTION ISSUE

The Issue signal, which moves instructions and their translations through U1 and U2 into the scoreboard, and the Skip signal, which moves these instructions up to U2, are controlled by the GO/Stop FF. Conditions necessary for these signals are as follows:

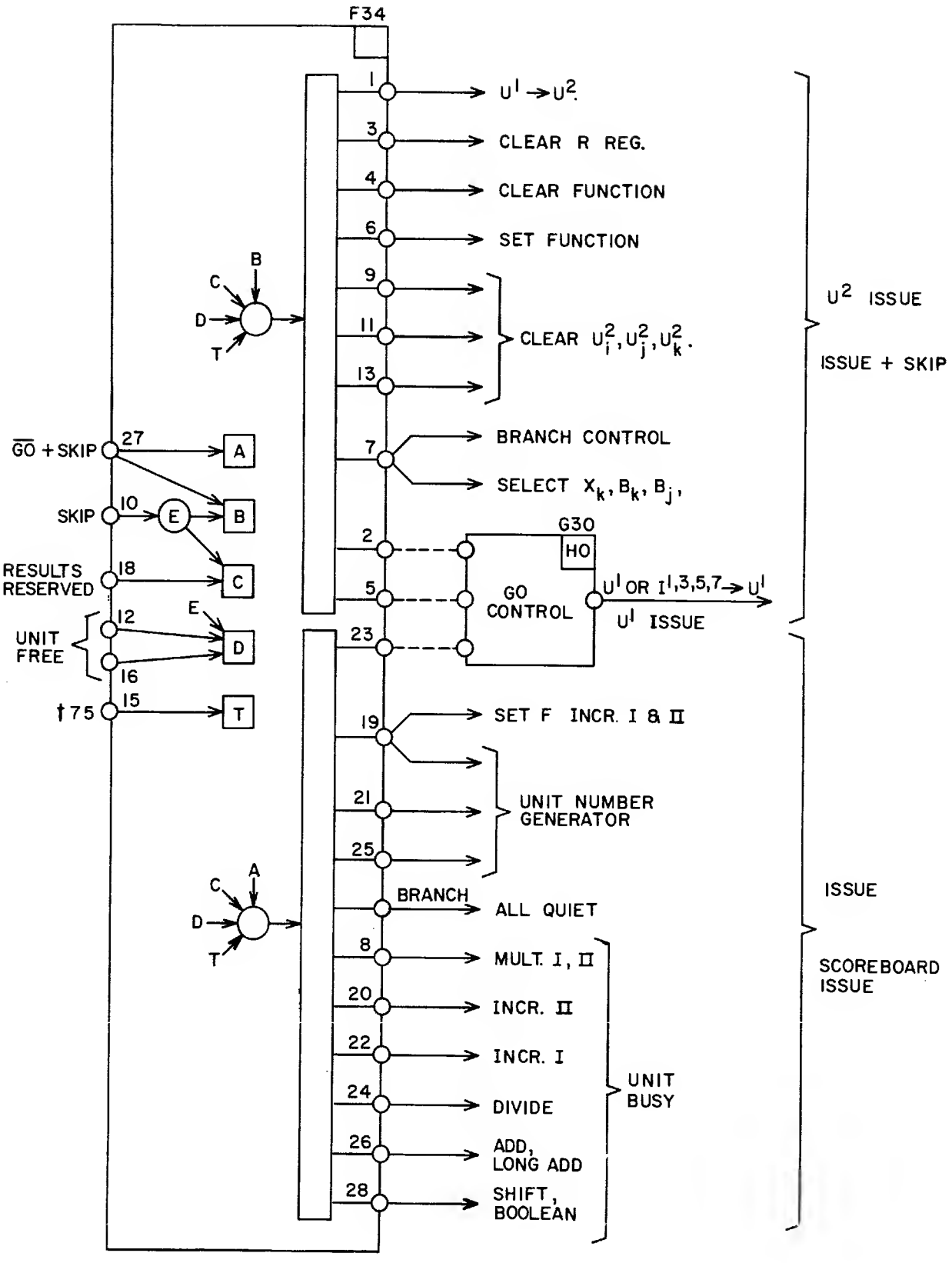


Figure 4-15. Issue Control

ISSUE: (Go)(Skip)(Unit Request)(Unit Free)(Destination Register Free)(t90)

SKIP: During the 200 ns after Proceed and during 100 ns after a 30-bit instruction where K is an operand.

The following section describes the conditions that place the Go/Stop FF in the Stop state. (Refer to Figure 4-16)

1) PAUSE

When the first parcel is extracted from the 60-bit instruction word in I0, the inch process begins. The beginning of the inch process, in turn, requests the next instruction word from central memory. When all parcels of the word in I0 (meanwhile, inching may have moved the word into I1) have been transferred to the scoreboard, the central processor waits for the next instruction word to arrive from memory. This wait period is called Pause.

To get the Pause signal, the inch process sets the Request Pause I FF (F37, TP5). When parcel 2 is taken from the stack, the Request Pause II FF is set (F37, TP6). The next time PK = 1, parcel 3 is placed in the scoreboard and issue is stopped. (Refer to Figure 4-16) The Issue and Skip signals are restarted by instruction ready control.

2) EXCHANGE JUMP STOP

During the Exchange Jump process, the Go/Stop FF is set to the stop state. The Exchange Jump signal, issued by a peripheral processor, forces the stop when PK = 1. (A PK = 1 count indicates parcel 3 was the last parcel issued or that no parcels have been issue the scoreboard. Thus, the issuance of an instruction word must be complete before permitting the stop for an Exchange Jump.)

3) ERROR STOP

The contents of bits 2^0 - 2^2 in the Exit mode register specify which errors can stop instruction issue. The Program FF (I7, TP6), cleared at the beginning of a new program by the Exchange Jump, insures that only the first error sets the Error Stop FF. The error signal is also gated by a Skip signal. The Skip signal gate permits finishing a loop in the stack (for branch cases) before stopping. (Only the next Branch instruction performs a Skip.)

The Error Stop FF is cleared by a master clear or by the execution of the Exit mode process.

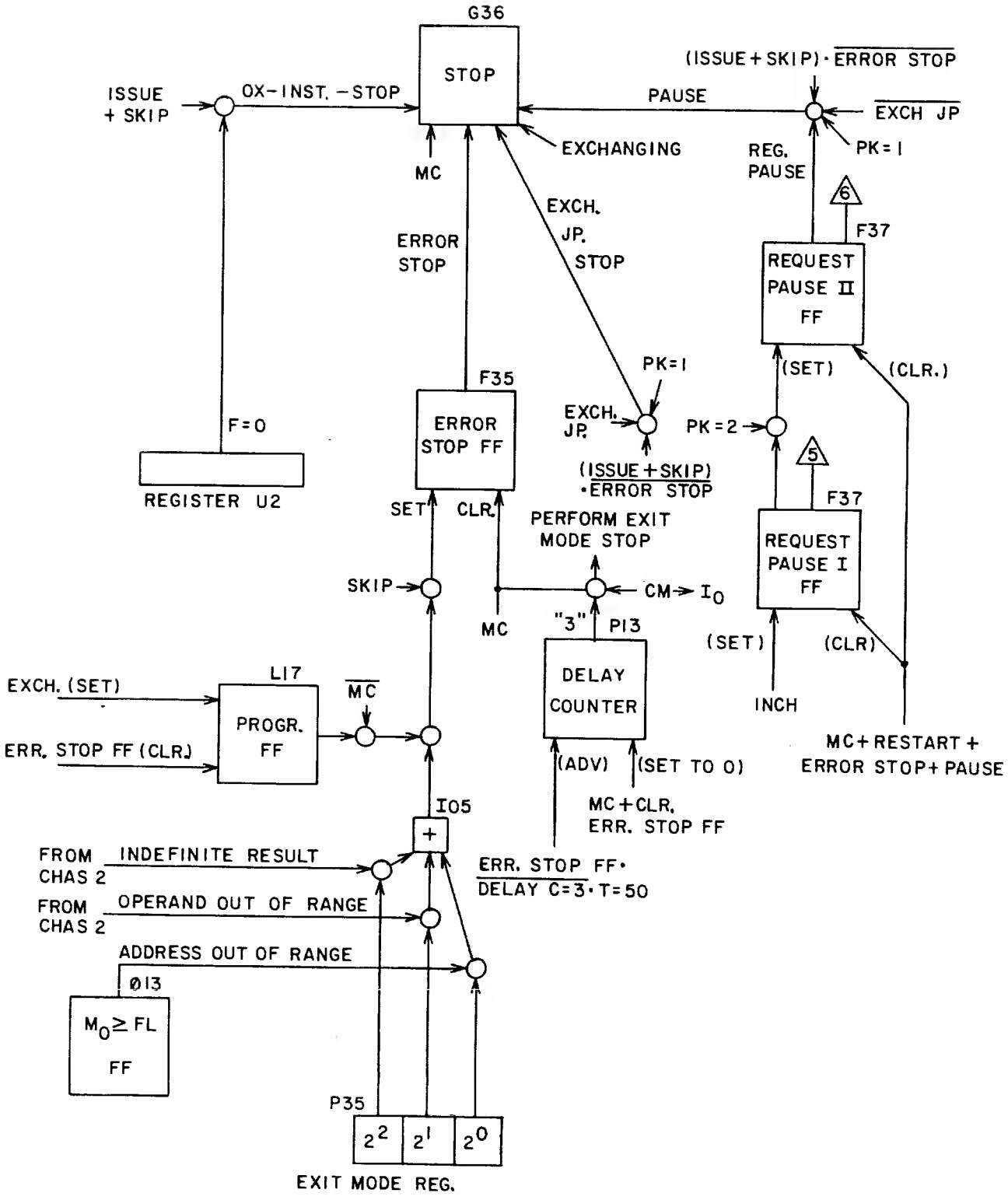


Figure 4-16. Stop Instruction Issue

4) OX - INSTRUCTION STOP

When a Branch instruction is issued into the scoreboard, further instruction issue halts to allow time for branch condition tests; i.e., branch/no branch, jump/loop, etc.

5) MASTER CLEAR

A master clear stops all instruction issues.

PROCEED INSTRUCTION ISSUES

The requirements for proceeding with instruction issues are outlined below:

1) INITIAL START OR PAUSE RESTART:

When the central processor starts (after an Exchange Jump), or proceeds (after referencing memory for the next instruction word; i.e., pause), two transfer pulses are required to bring the first parcel (parcel 0) from IO into U2. No information is placed in the scoreboard during this period; these transfer pulses that move the parcel to U2, but not to the scoreboard are called "Skips".

2) PROCEED AFTER A BRANCH INSTRUCTION:

Three possible "proceed" cases exist after a Branch instruction:

a) $\overline{\text{BRANCH}}$: When the branch condition is not met, the next instruction may be in the same 60-bit instruction word as the Branch instruction.

b) LOOP: The branch condition is met, and the next instruction is parcel 0 of an instruction word in the stack.

c) JUMP: The branch condition is met; the next instruction must be obtained by a memory reference.

Case (a), if the Branch instruction was in parcel 0 or 1, requires one parcel be skipped. In cases (b) and (c), where a new instruction word is requested, it is necessary to perform two skips to transfer the new instruction into U2.

For simplicity, the logic treats all "proceeds" after a stop condition the same (i.e., as requiring two skips). Case (a) above, the only case requiring only one skip, is

reverted to the normal case by disabling one advance pulse of the parcel counter. Thus after every stop condition, two skips are performed before information can be issued to the scoreboard.

3) In cases (1) and (2c), which require an instruction word from memory, a "proceed" signal occurs only if the new instruction word is available in IO.

OPERATION

A Proceed signal may originate from one of three different sources (cases): (Refer to Figure 4-17)

1) Restart (Instruction Ready Control)
2) Loop
3) Branch

RESTART

The Restart signal occurs when a new instruction word is available in IO. This occurs in the following cases:

a) End Exchange (start after an Exchange Jump)
b) Pause (stop between instruction words)
c) Jump (Branch is not in the stack)

The corresponding signal sets the CP Stopped FF, clears PK, sets L = 7 and (in the jump case) D = 7, and sets the Enable Restart FF. (Refer to Figure 4-18)

When the 10-tag (RNI tag) is detected in the hopper and the address is accepted by central memory, the new instruction word is read from memory. The new instruction word is guided to IO, and the Instruction Available FF is set, permitting the Restart. Note that it is possible for an instruction to be available before the Enable Restart FF is set. For example, a Functional Unit conflict before Pause, or (in the jump case) when a requested instruction word must be in the stack before the Go Branch signal occurs.

Loop

If the Branch test indicates that the next instruction is located in the stack, the Proceed signal is issued. (In case the inch process is occurring, the Inching FF delays the Go Branch signal until movement in the stack is completed.)

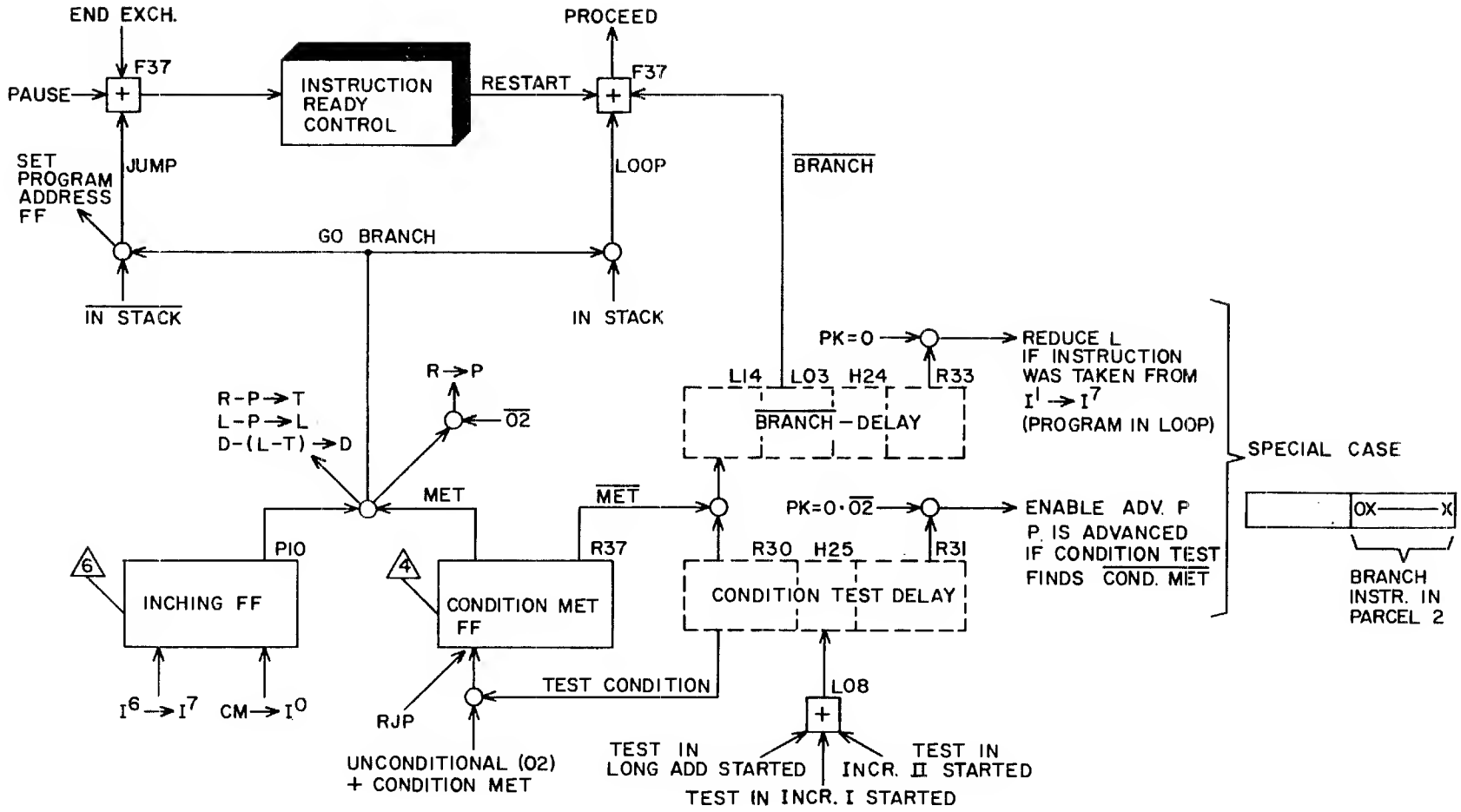


Figure 4-17. Proceed Instruction Issue

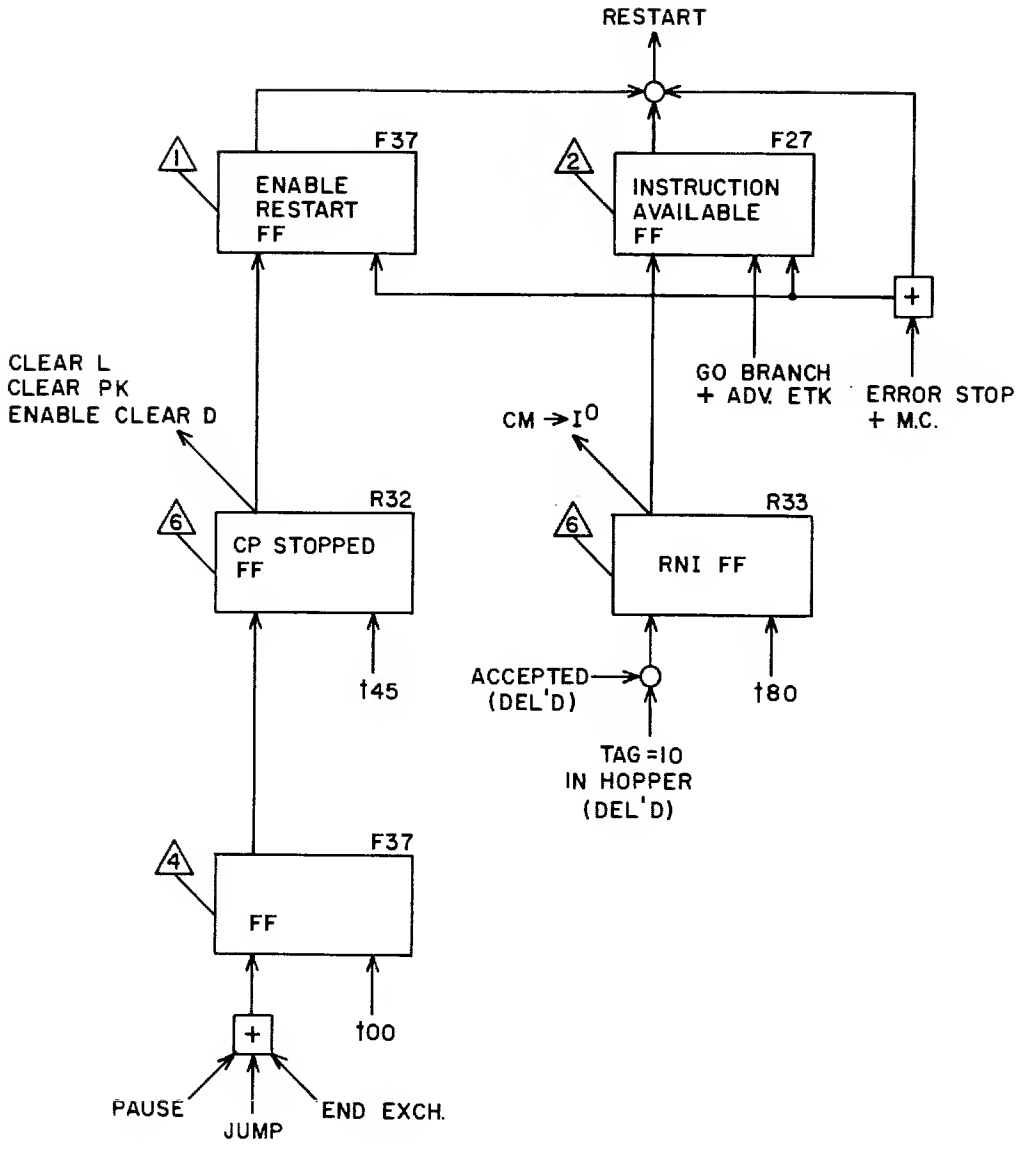


Figure 4-18. Instruction Ready Control

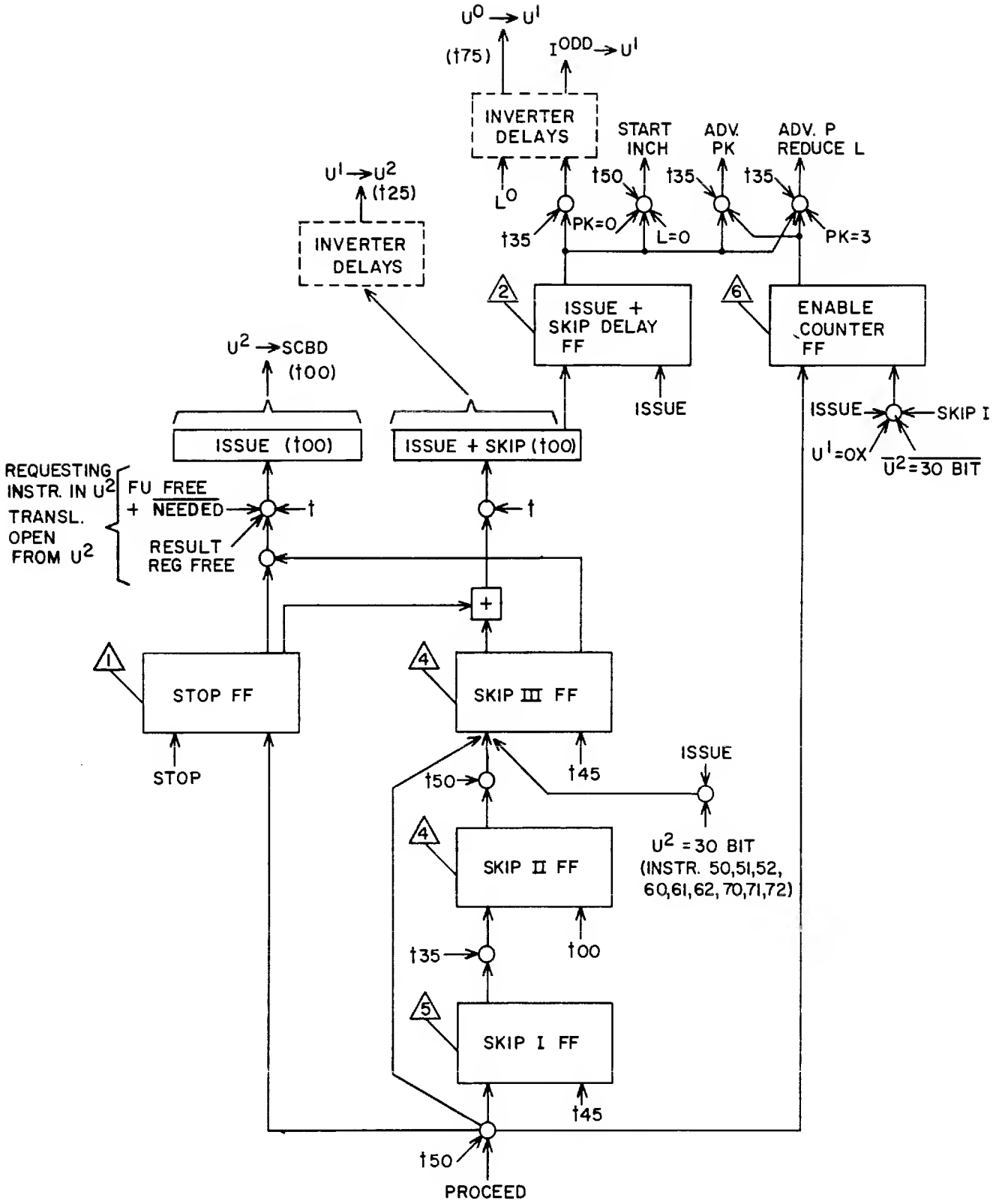


Figure 4-19 Go Control/Issue/Skip

NO BRANCH

During a conditionl Branch instruction, if the branch condition is not met, the Proceed signal can be issued. (As previously stated, the parcel counter skips one advancement in Branch instructions; this permits two skips following a Proceed signal.)

A special case exists for a $\overline{\text{branch}}$ condition. During a Branch instruction, the P and L counter are disabled. This disabling occurs from the moment a Branch instruction is detected in U1 until the Proceed signal is issued. (Meanwhile, the parcel counter is advanced by one.) If a Branch instruction comes from parcel 2 of an instruction word and enters U2, the parcel count is now 3. Normally on PK = 3, P is advanced and L is reduced. However, in Branch instructions this does not occur. Action proceeds with a PK = 0; PK is counted to 2 during the two Skips, then a stop occurs to wait for the new instruction word. Since this action would not provide opportunity to correct the counts in P and L, special circuits exist to accomplish this before the Proceed signal is given.

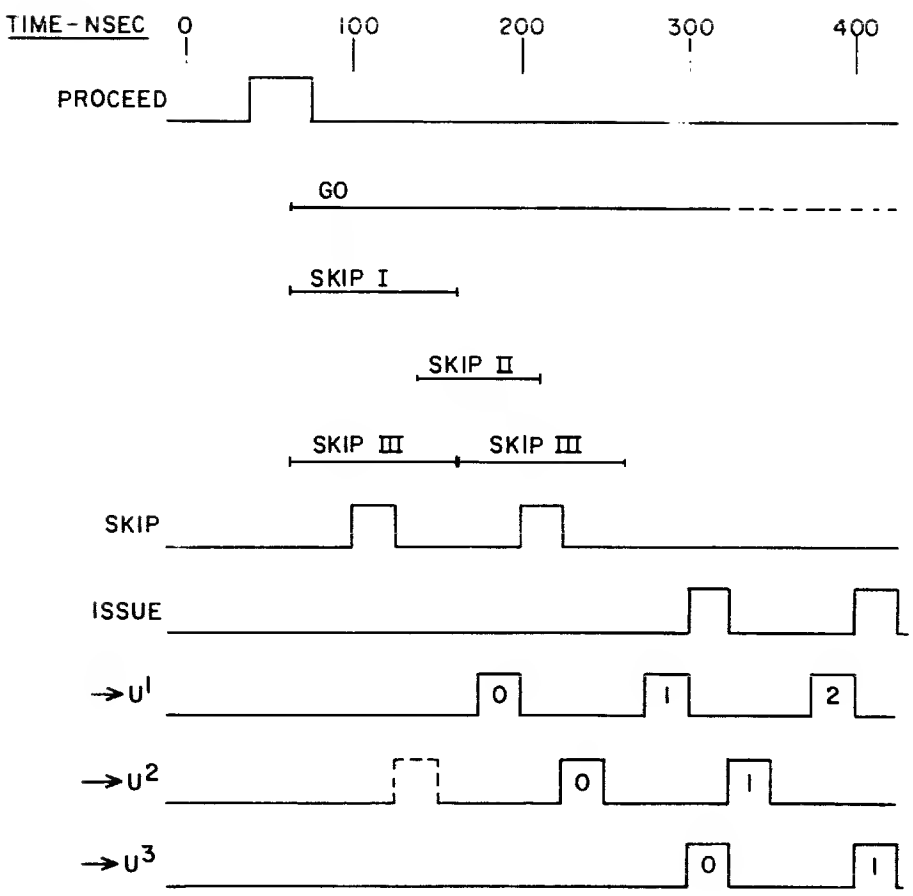
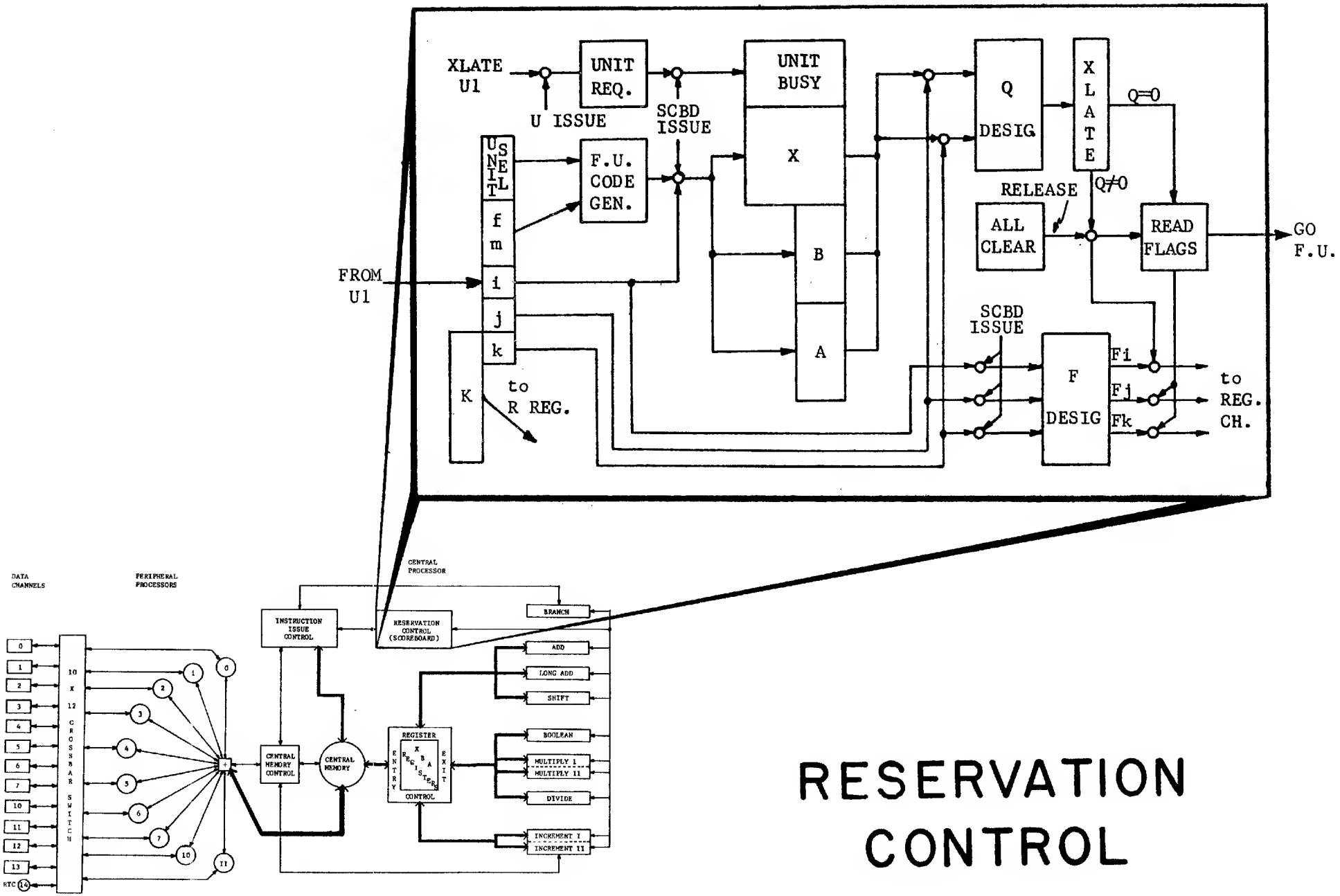

TIME - NSEC
0
100
200
300
400
PROCEED
GO
SKIP I
SKIP II
SKIP III
SKIP III
SKIP
ISSUE
→U1
0
1
2
→U2
0
1
→U3
0
1

Figure 4-20

CHAPTER V

RESERVATION CONTROL



RESERVATION CONTROL

CHAPTER V

RESERVATION CONTROL

INTRODUCTION

The scoreboard directs the exchange of operands and results among the 24 operating registers, central memory, and the 10 functional units. Instructions are issued to the units for execution in the order prescribed by the original program sequence. The scoreboard permits instructions to be executed out of order while retaining the original program sequence.

Scoreboard reservations are made in serial order for one unit at a time and at a maximum rate of one every minor cycle; this corresponds to the maximum instruction issue rate. However, processing of requests to read operands or store results from all units goes on in parallel and coincidently with placing of reservations. Hence, program instructions are executed in a series-parallel arrangement resulting in very high-speed running of a program.

The scoreboard uses a reservation system on the registers and units to direct the instruction issue and execution sequence. The scheme allows all units to be in operation at the same time but prevents one unit from executing more than one instruction at one time or more than one unit from storing in a common register at the same time. Lockouts for the functional units and operating registers and a series of designators which identify registers and units aid the control system.

An instruction is issued to a functional unit only if the unit and the required result register are free; i.e., they are not reserved for another instruction or another result. With both elements free, the instruction is issued and reservations placed in the scoreboard for the necessary operating registers and unit (Figure 5-1).

Scoreboard control directs the unit in obtaining its operands and storing its result; computation in the unit proceeds independently. The unit requests permission from the scoreboard to release its result to its result register. The scoreboard determines that the path to the register is clear and signals the requesting unit to release its result. The releasing unit's reservations are then cleared and units waiting for the result are signalled to read the result for their computation.

Special scoreboard action handles the reservation and control scheme for address modification in the increment units. A change to an A1-A7 address register changes the corresponding X1-X7 operand register so that an operand is read into X1-X5 or the content of X6-X7 is stored. Hence, for 50-57 instructions, the A result register and associated X register are reserved together to prevent

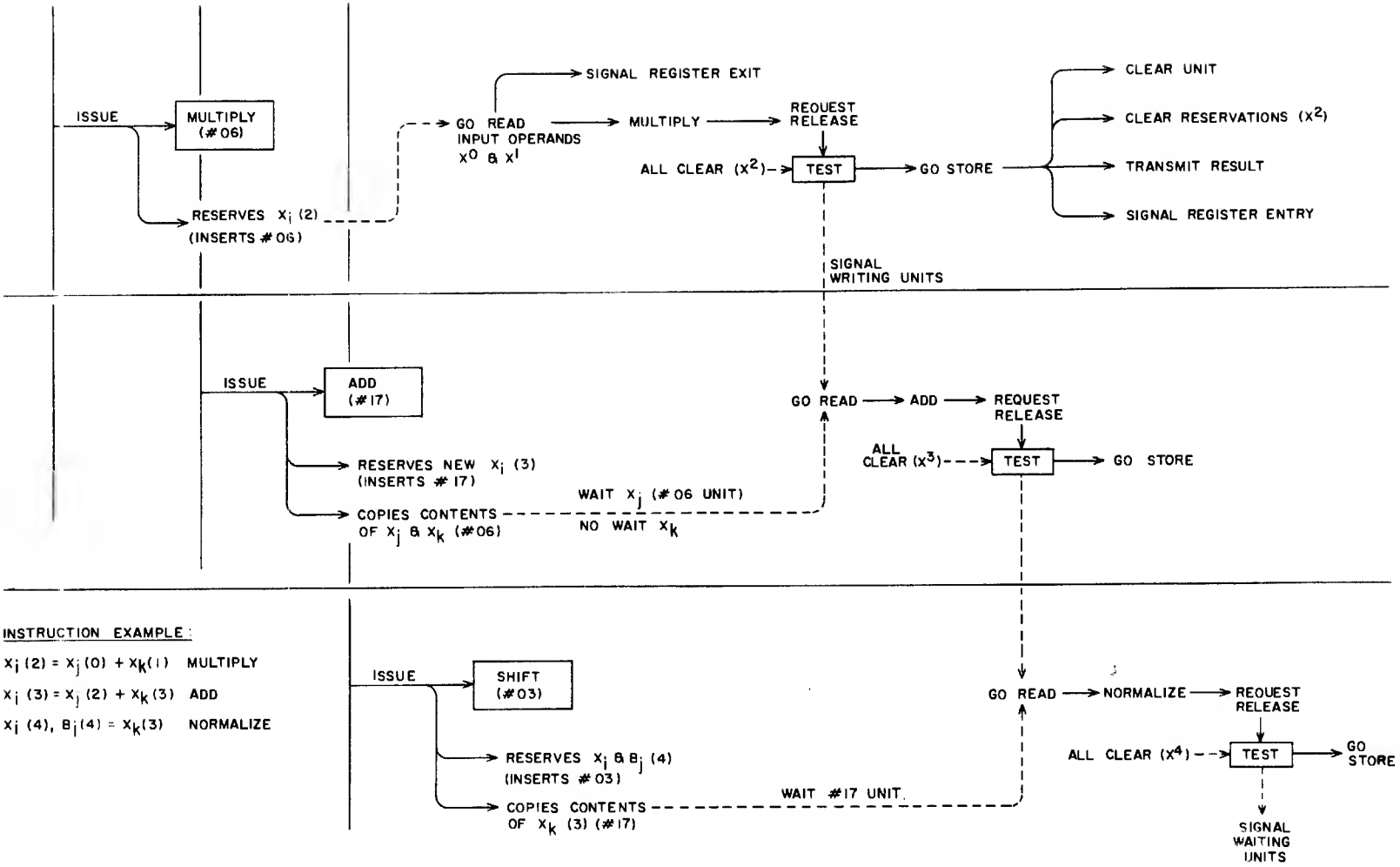


Figure 5-1. Scoreboard Control

the X register from being pre-empted by a subsequent instruction.

The increment unit computes the address and sends it to the stunt box for issue to CM. Coincidently, the result is released to the A register through normal scoreboard processing. Also, the releasing increment unit's reservations are cleared to free the unit for further computation. However, the X register reservation is held until after the address is accepted by CM (thus spanning any delay time in the stunt box because of bank conflict) and the data word is delivered to the X register (for X1-X5 cases) or delivered to the CM store distributor (for X6-X7 cases). Additional information is given later for the increment and branch unit action in the scoreboard.

The functional units exchange data with the 24 operating registers over a number of data trunks. A priority system regulates trunk usage, and the scoreboard selects the proper data trunk and honors the priority system in directing the data exchange.

The designators in the reservation system are listed below and shown in Figure 5-2.

X	Operand register	j	First entry operand register designator
B	Increment register	k	Second entry operand register designator
A	Address register	i	Result register designator
F	Functional	Q	Entry operand reservation designator

Note that reservation designators are part of the scoreboard circuits but are shown in Figure 5-2 as related to the entry operand and result registers of a given unit.

Scoreboard operation is described first from the view of placing reservations, and second, the control system which interprets reservations and directs a unit to read its entry operands and store its result.

PLACING RESERVATIONS

Reservations are placed in four sequential steps following issue to the scoreboard.

1. Make functional unit reservation (Set Unit Busy).
2. Assign operating registers to functional unit (set F).

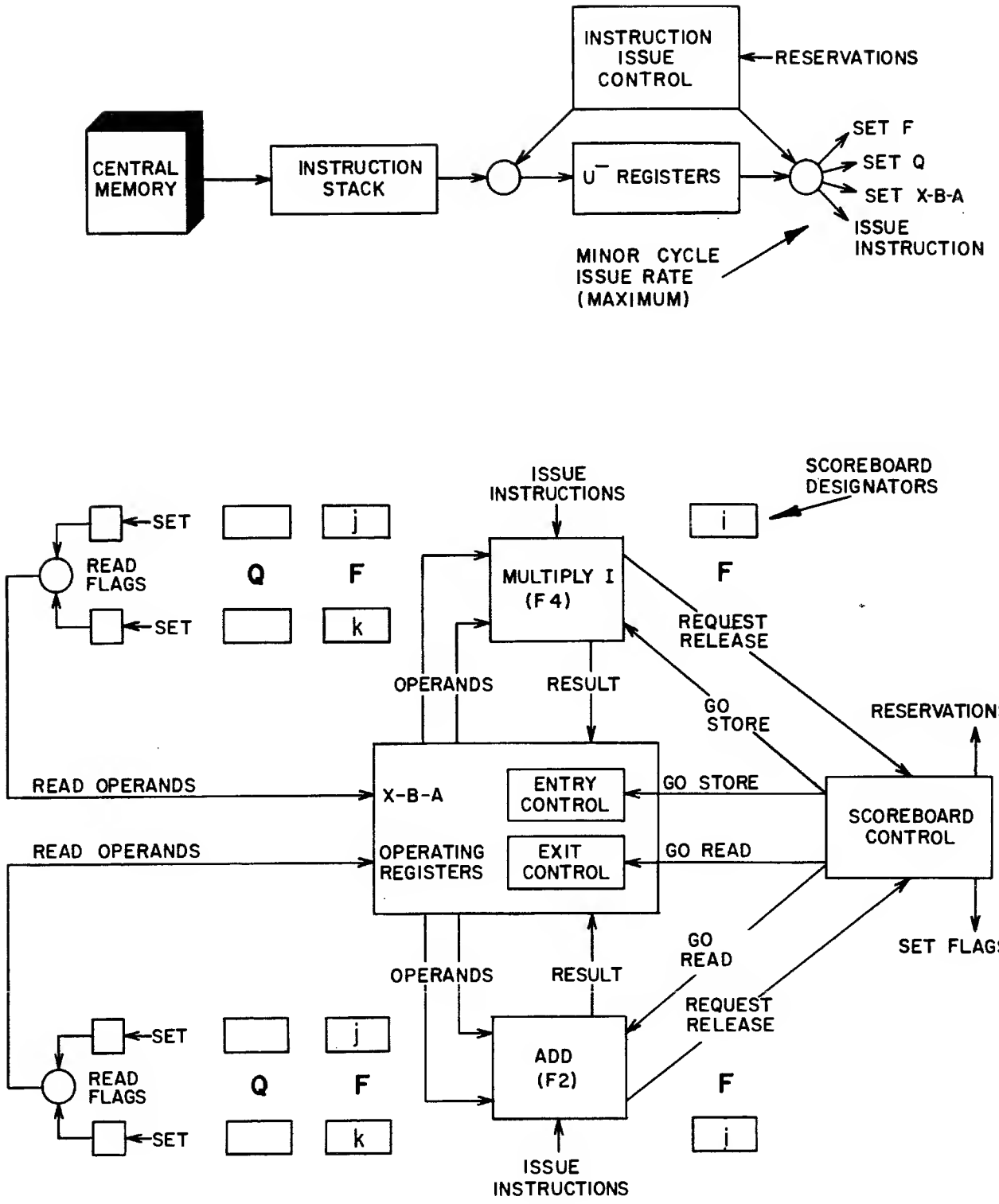


Figure 5-2. Reservation Designators

3. Determine previous reservations on entry operand regs. (set Q).
4. Update reservation list to reflect the result register of the latest reserving unit (set XBA).

UNIT BUSY

The fm portion of the instruction is translated at the output of U1 selecting one of the following Unit Request FFs:

a) Shift, Divide, Add, Long Add, Boolean (G37, H36)
The outputs of these FFs are sent to their respective Unit Busy FFs, (F27, 29, 30). The input gates of these cards, however, can only be made if the Unit Busy FF is in the clear state. If this gate is made we will enable the D gate on F34 of the Issue Control.

b) Multiply, Increment (H36)
As there are two Multiply and Increment units, these unit request FFs can set one of two Busy FFs under the same conditions described in paragraph a). To set Mult. II or Incr. II Busy FF, their respective unit I Busy FFs must be set.

For 5X instructions when i=0, only FF CD (Write or Increment) will be set on G26 and G27. FF AB (Read or Write) remains cleared, because no memory reference can be made by changing A0.

c) Memory Read, Memory Write (Ø19)
A 5X translation and i=6+7 will set the Memory Write FF on Ø19. A 5X translation and i=1-5 will set the Memory Read FF. These FFs in turn can set the Busy FFs for increment I or II on G26 if all conditions are satisfied.

d) Pass (H36)
This FF on H36/TP6 will send its output directly to the Issue Control to enable gate D on F34.

F DESIGNATORS

The set F step sets the result and entry operand register designators i, j, k for the selected unit. This assigns operating registers to the selected unit. The i designator identifies a result register for most instructions and j and k entry operands. Each 3-bit designator specifies one of the eight registers in an X, B or A register group. For example, the j designator of the add unit may be set to X1, which reserves operand register X1 as one of the entry operand registers.

Table 5-1 shows that for the Shift and Increment Units, the F designators can specify X, B or A registers. For these units therefore, a decision must be made as to which register F designator must be set for a particular instruction.

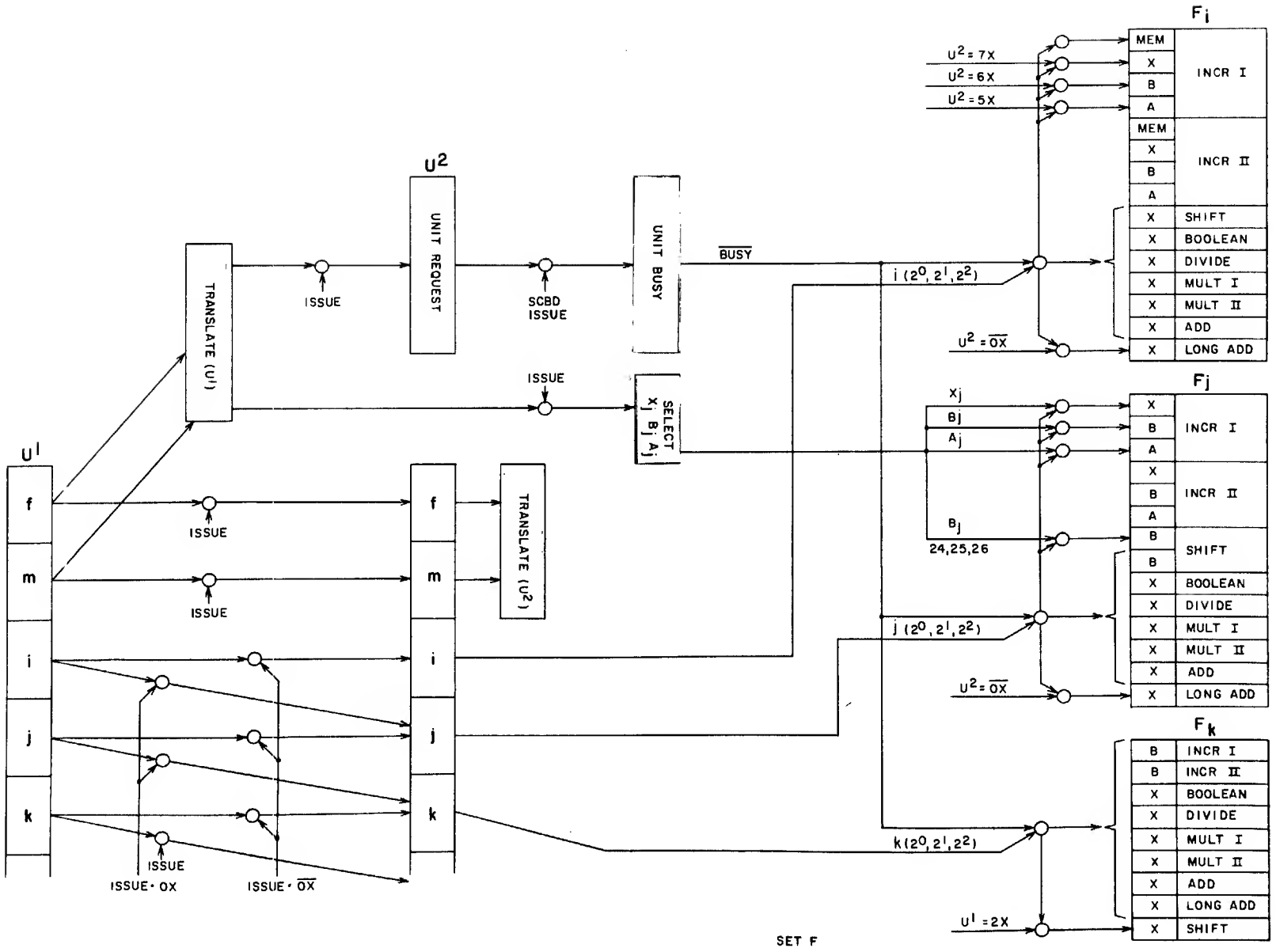

U1
f
m
i
j
k
TRANSLATE (U1)
ISSUE
U2
UNIT REQUEST
SCBD ISSUE
UNIT BUSY
BUSY
ISSUE
SELECT Xj Bj Aj
ISSUE
ISSUE
f
m
i
j
k
TRANSLATE (U2)
ISSUE
ISSUE · OX
ISSUE · OX
Fi
U2 = 7X
U2 = 6X
U2 = 5X
MEM
X
B
A
INCR I
MEM
X
B
A
INCR II
X SHIFT
X BOOLEAN
X DIVIDE
X MULT I
X MULT II
X ADD
X LONG ADD
i (20, 21, 22)
U2 = OX
Fj
Xj
Bj
Aj
X
B
A
INCR I
X
B
A
INCR II
Bj
24,25,26
B
B
SHIFT
X BOOLEAN
X DIVIDE
X MULT I
X MULT II
X ADD
X LONG ADD
j (20, 21, 22)
U2 = OX
Fk
B INCR I
B INCR II
X BOOLEAN
X DIVIDE
X MULT I
X MULT II
X ADD
X LONG ADD
X SHIFT
k (20, 21, 22)
U1 = 2X
SET F

Figure 5-3

UNIT	F_j			F_k		F_i		
	X	B	A	X	B	X	B	A
ADD	X			X		X		
MULTIPLY I	X			X		X		
MULTIPLY II	X			X		X		
DIVIDE	X			X		X		
LONG ADD	X			X		X		
BOOLEAN	X			X		X		
SHIFT		X		X		X	X	
INCREMENT I	X	X	X		X	X	X	X
INCREMENT II	X	X	X		X	X	X	X

Table 5-1

This selection is done by the U2 translation and the select Xj, Bj, Aj FFs which will enable the required F designator or designators.

Q DESIGNATORS

The set Q step determines if the entry operand registers are reserved for results of other units already in operation, and which units have them reserved. The fact that the current instruction has been issued to a unit indicates that its own result register is not reserved. The register reservation information is held in 24 separate X-B-A designators. Each unit is assigned a number for reservation purposes (table 5-2) and the number of the last reserving unit is transferred from the proper X-B-A designator to the Q designator of each entry operand of the waiting unit.

For example, if register X1 was already reserved for the result of Multiply 1 unit and the Add unit wants to use X1 as its j entry operand, then the Q_j designator of the add unit will be set to the unit code (06) identifying the multiply 1 unit. (See table 5-2) If the required register is not reserved, the respective Q designator will be set to zero and its Read flag will be set to indicate that the register is free and ready to be read. However, a unit will read its operands only when both Read flags are set.

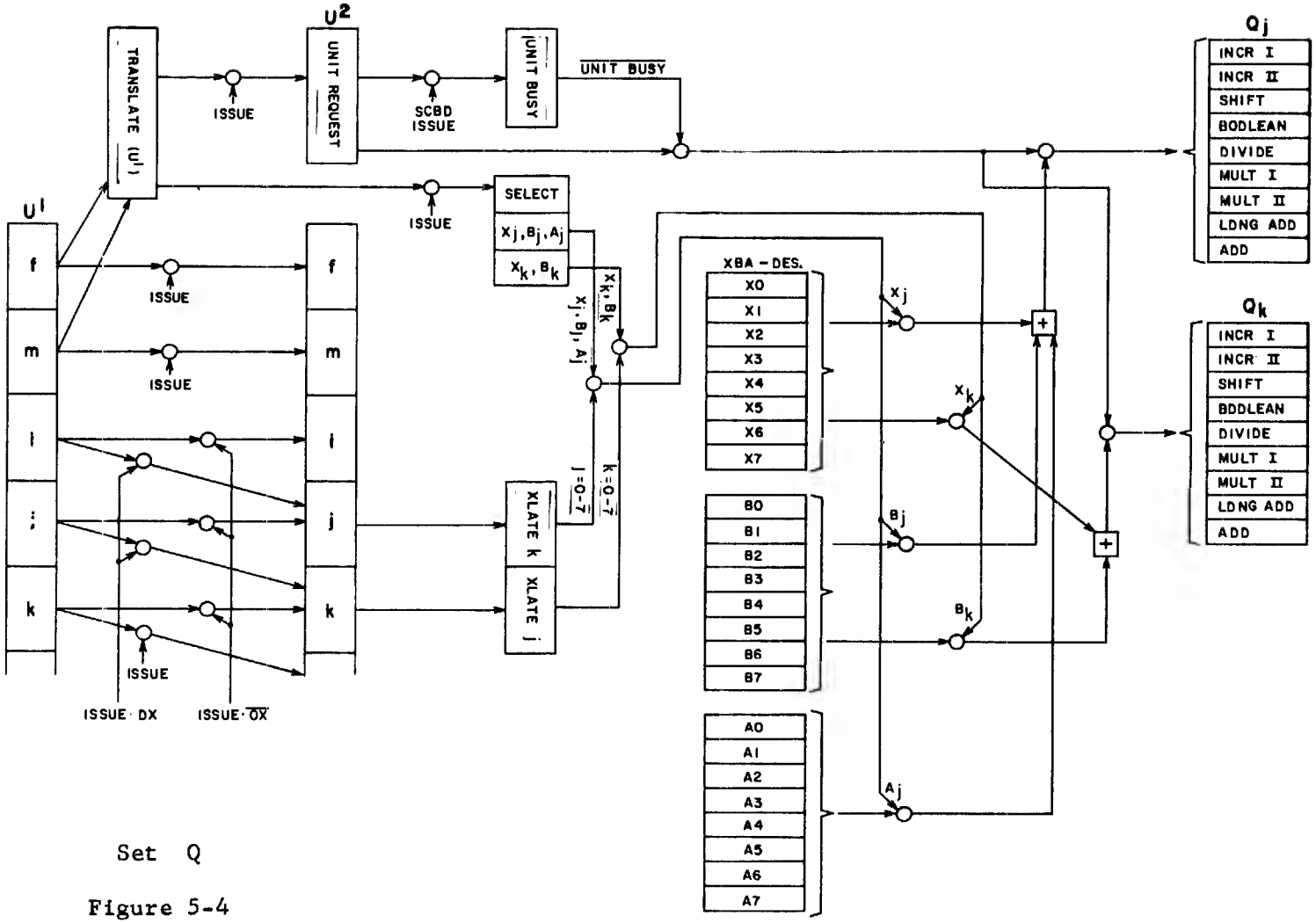
Q designators are set in the following manner:

After setting the Unit Select FF, the entry operand register numbers for the requesting unit are identified from translation of the i, j, k portions of U2. Select FFs, corresponding to that instructions entry operands (X_j, X_k, B_j, etc.), are set to identify the XBA register groups. In the case of an add unit selection, X_j and X_k FFs (H26 and F31) are set and their outputs are combined with the k and j translation from U2 (G22, G23) to identify and read the respective X designators (H01-08). This information is sent to the Q designators (I17, I18) of the add unit under control of the add unit select FF.

Q Designators for Functional Units

Q (octal)	Functional Unit
00	Branch
01	Increment 1
02	Increment 2
03	Shift
04	Boolean
05	Divide
06	Multiply 1
07	Multiply 2
10	----
11	Read Memory Channel 1 (X1)
12	Read Memory Channel 2 (X2)
13	Read Memory Channel 3 (X3)
14	Read Memory Channel 4 (X4)
15	Read Memory Channel 5 (X5)
16	Long Add
17	Add

Table 5-2



Set Q

Figure 5-4

XBA DESIGNATORS

The set XBA step updates the XBA reservation list to include the result register of the latest or current reserving unit. In the

identifying code for the add unit (octal 17) is entered in the X0 designator as the latest reservation. Should a subsequent instruction require X0 as a result register, it is not issued until the add unit releases X0. Instructions which require X0 as an entry operand however, are issued and receive the add unit code in their respective Q designator.

Three steps are necessary to place this reservation:

1. The unit is identified
2. The register number is identified
3. The register group (XBA) is identified

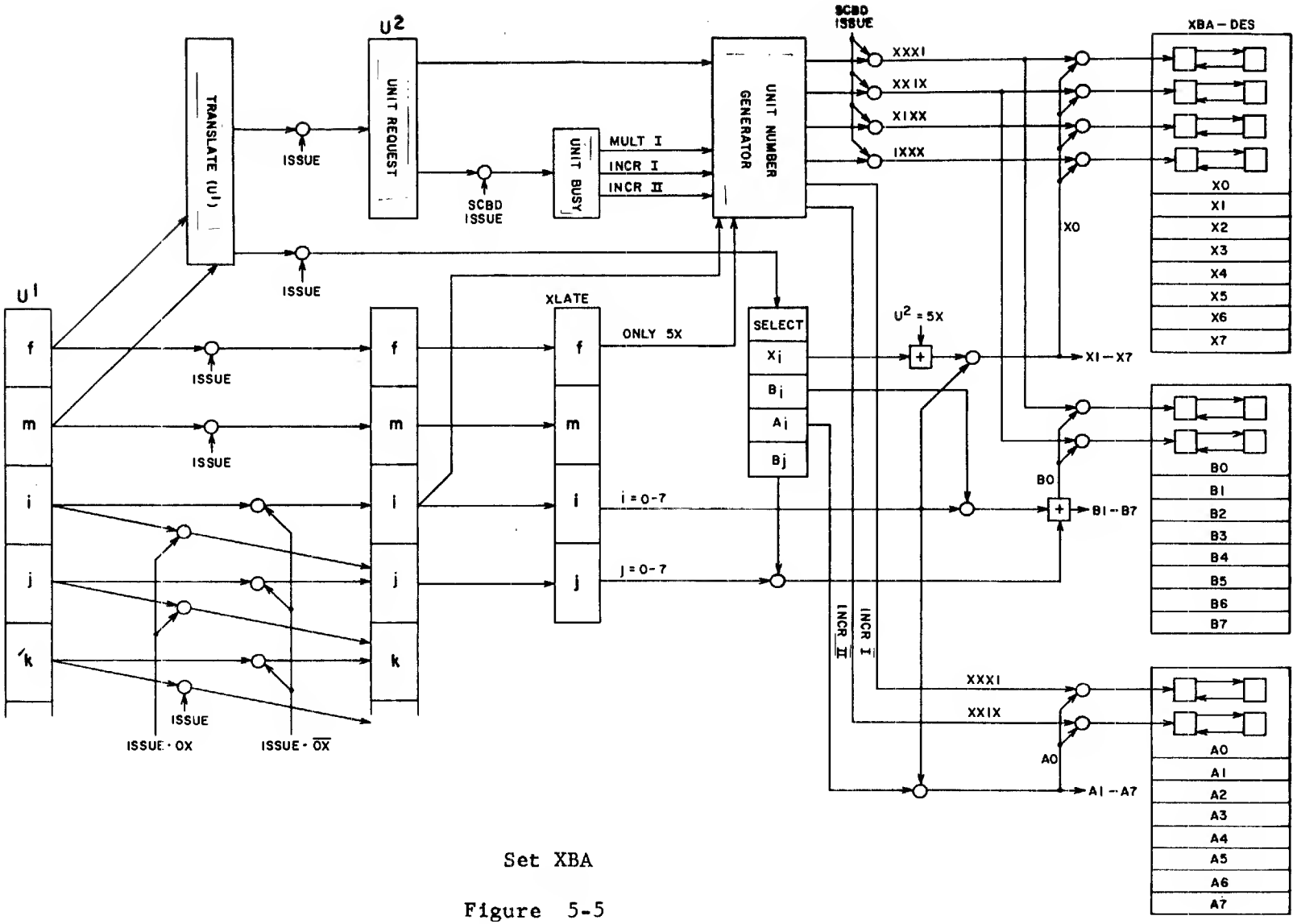
The unit is identified by translating the fm portion of U1; this will set the unit select FF. A second translation of the fm portion of U1 identifies one of four possible result register groups: Ai, Bi, Xi, or Bj. This information is stored in separate Result FFs.

The register number is derived from translations of the j and k portions of U2. The register number and group (XBA) are combined to gate the unit code (Table 5-2) into the XBA designator. Each unit selection is converted to a 4-bit binary code by driving four inverters in a combination equalling the binary code. Thus, the add selection drives four inverters which yield binary 1111 or octal 17; a shift selection, for example, drives only bits 0 and 1 to yield binary 0011 or octal 03. (See G33 and F40.) The shift and increment units are the only ones that use the A and B registers for results. The codes for these units use only bit positions 0 and 1. Thus, the B and A designators need be only two bits, whereas the X designators are four.

5X instructions place their result in an A register and thereby cause a read or write operation in the corresponding X register (except for AC and X0). In these cases, both registers are reserved.

SET READ FLAGS

When the Q designators have been set, the Q translators will provide outputs for Q=0 through 17 (Q=10 not used). These octal codes define the unit that has reserved the particular register. (See Table 5-2) For example, let Q=01. The Increment 1 unit has reserved the particular register referred to by the Q designator. The Q=01 output from the Q translator will be anded with the output of the Increment 1 Release FF setting the operand read flag. In the



Set XBA

Figure 5-5

case of Q=0 the read flag is set directly, since that particular register had not been reserved by another unit. Operands are read only when both read flags are set.

RESERVATION CONTROL

A unit reads its operands, computes, and stores its result after its reservations have been placed. Scoreboard control directs the read operand and store result action. The example used to describe placing reservations is described further to illustrate control operation.

In the example, an add instruction is issued to the add unit and the necessary register reservations made. Before this, a multiply instruction was issued to the multiply 1 unit, and this unit reserved register X1 for its result. The add unit needs X1 for its j operand. The add unit must wait for the multiply 1 result and release of X1 before it can read X1. Scoreboard control requires that both operands be available before a unit proceeds to fetch them (discussed later).

Multiply 1 unit, when ready, requests scoreboard permission to release its result to X1. Control determines that the data trunk to X1 is all clear and signals multiply 1 to go store its result in X1. Control then voids all multiply 1 reservations by clearing the multiply 1 busy FF and the X1 designator in the X-B-A reservation designators.

At the same time, all Q designators in the scoreboard are tested to determine if any units are waiting for the result being stored in X1. For our example, the test finds the add unit j operand waiting for X1 and sets a corresponding read flag to indicate that the j operand is now ready. The k operand has a similar flag. Assuming the k operand read flag set (because its Q=0, thus no reservation conflict), and the data trunk from the add unit to the operating registers clear, control signals the add unit to go read its operands. The add unit reads its operands, computes, and finally requests scoreboard permission to release its result to X0. The request release sequence described earlier for multiply 1 then repeats. Note that several units may be waiting for X1 and may read X1 coincidently. The number of simultaneous reads of a given register relates to the assignment of data trunks from the operating registers to the units (Table 5-5).

Release requests from units arrive at scoreboard control asynchronous to other scoreboard action. Requests are immediately recognized and processed in parallel with each other and other scoreboard action. As noted, units share a number of data trunks connecting them to the 24 operating registers. A priority scheme on the data trunks regulates their use and may produce a small delay in processing go store and go read commands. Otherwise, parallel processing is

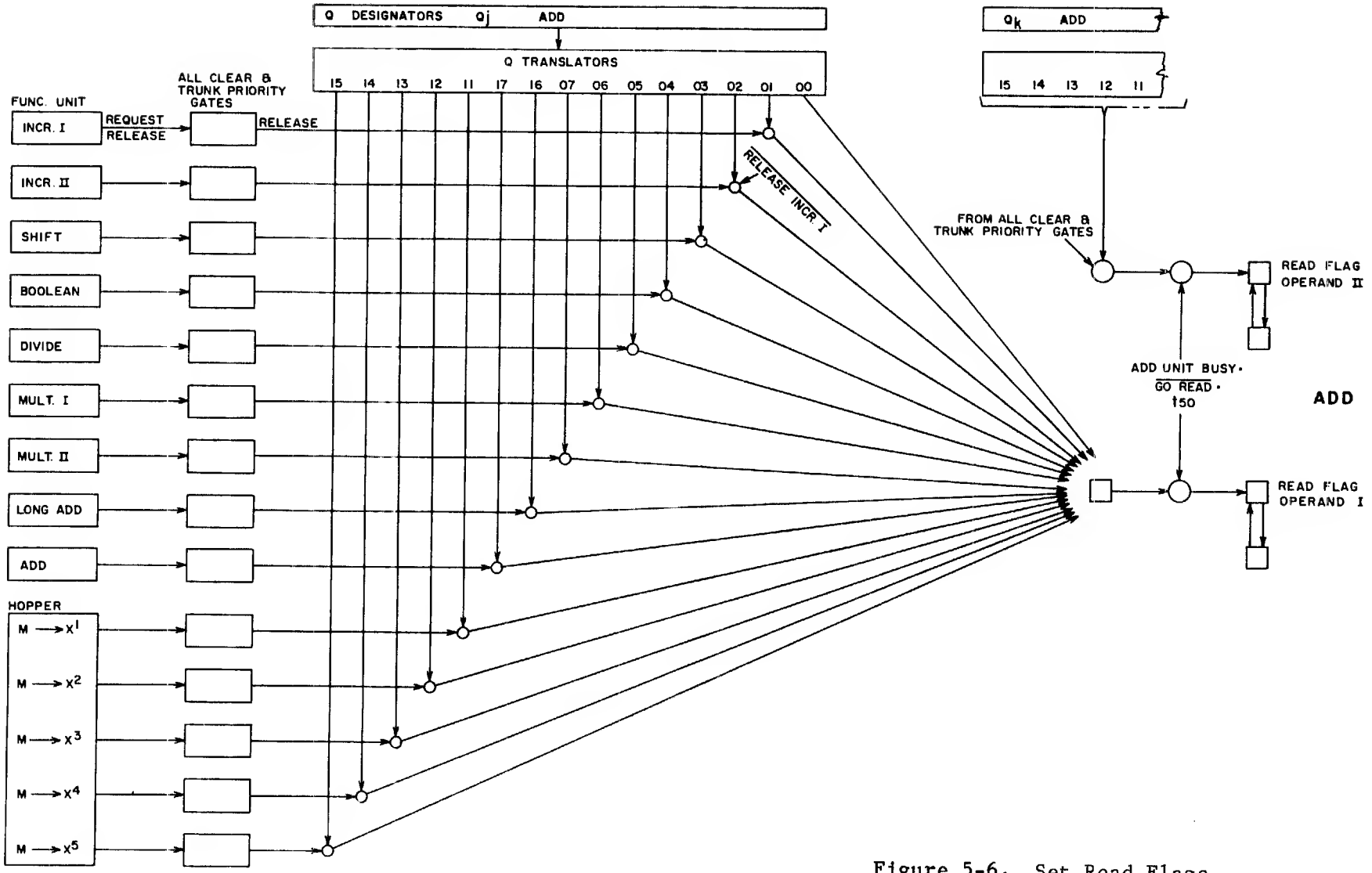


Figure 5-6. Set Read Flags
Add Unit

as long as instructions are issued.

RELEASE

Circuit discussions begin with the unit "request release" command and continue with the scoreboard "go store" and "go read" commands and the circuits which control their production.

REQUEST RELEASE

The "request release" from a unit indicates the unit wants to release its result to an operating register. The command is sent several minor cycles before the result is actually ready; the premature command allows scoreboard control to process the request and determine the next in-line user(s). Timing permits a unit to store its result and the next user(s) to read the result at a slightly later time. The overall timing scheme from "request release", to a store and then a read on a register, is such that one minor cycle elapses from the end of one unit's computation and the start of a unit which is waiting for the result.

"Request release" signals are stored in separate FFs to allow parallel processing. Each "request release" is transferred through an all clear gate and a trunk priority gate to a release FF. If these gates can be made, the Release FF will set, sending a command to the unit to transmit its result. The next user is also determined at this time and told to go read its operands.

ALL CLEAR

The "all clear" gate gives all units (which have been issued instructions) the opportunity to read a register before another unit stores its result in the register. This read-before-store condition arises because one or more instructions which call for reading a given register may be issued before an instruction which calls for storing in the same register. However, varying execution times of the units (and other factors, for example trunk priority) permit the unit executing the last instruction issued in a series to be ready to store its result before previously issued instructions have finished reading the register. The all clear resolves this conflict.

Earlier discussion pointed out that each entry operand for a unit had a read flag which, when set, indicated that the register associated with the operand was ready to be read. Both entry operand registers must be ready and their flags set before reading starts (flag set conditions are described later). Hence, one set flag for a unit indicates that the unit is waiting for its other operand register to become available; both flags set indicate reading in process.

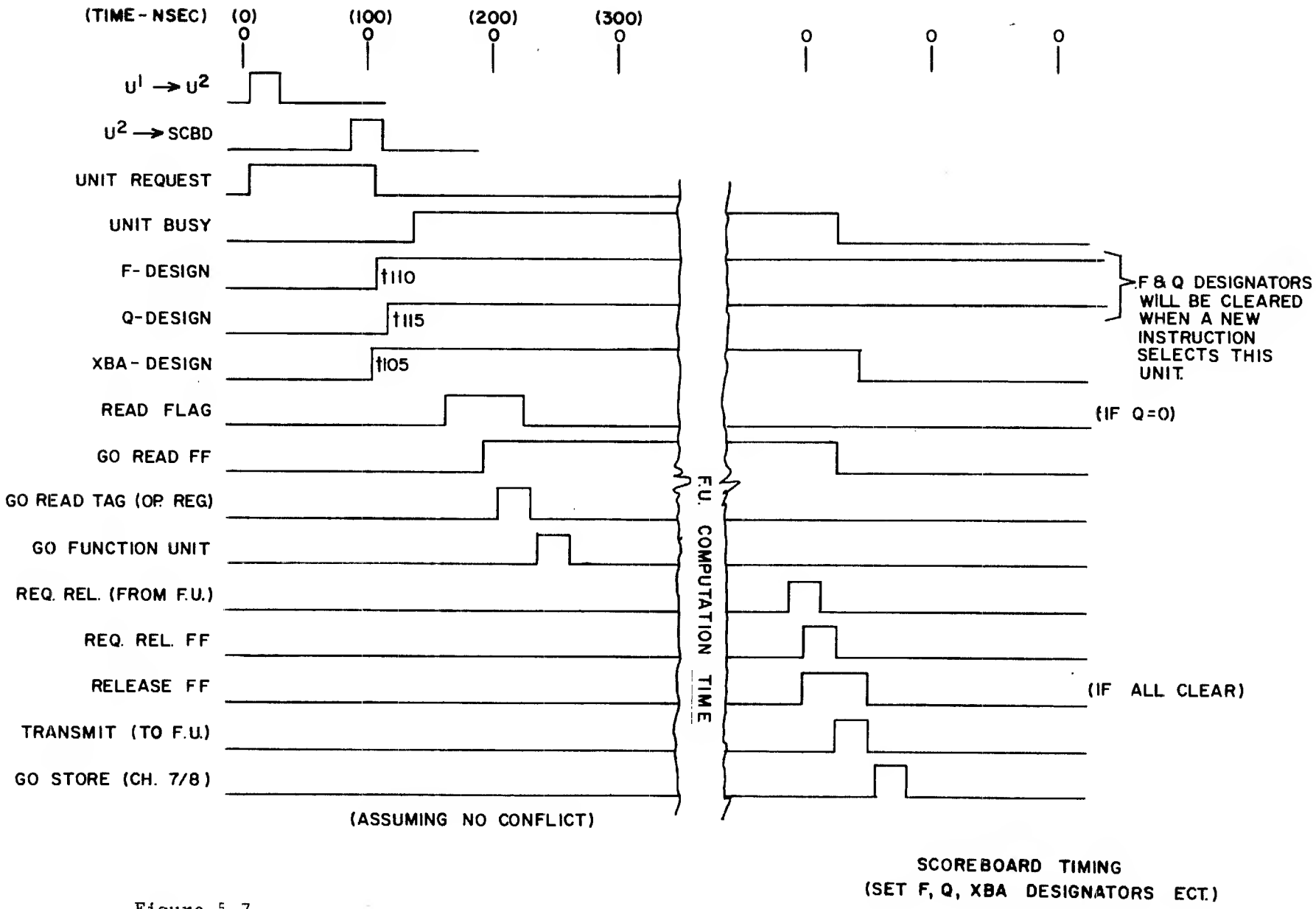

(TIME - NSEC)
(0)
(100)
(200)
(300)
U1 → U2
U2 → SCBD
UNIT REQUEST
UNIT BUSY
F-DESIGN
Q-DESIGN
XBA-DESIGN
READ FLAG
GO READ FF
GO READ TAG (OP. REG)
GO FUNCTION UNIT
REQ. REL. (FROM F.U.)
REQ. REL. FF
RELEASE FF
TRANSMIT (TO F.U.)
GO STORE (CH. 7/8)
†110
†115
†105
F.U. COMPUTATION TIME
F & Q DESIGNATORS WILL BE CLEARED WHEN A NEW INSTRUCTION SELECTS THIS UNIT.
(IF Q=0)
(IF ALL CLEAR)
(ASSUMING NO CONFLICT)
SCOREBOARD TIMING
(SET F, Q, XBA DESIGNATORS ECT.)

Figure 5-7

Either case voids the all clear condition and delays release.

The all clear condition tests read flags against the register receiving a result. Some units, like the add, communicate only with X registers and in these cases read flags related to A or B registers are not tested. However, the increment units may communicate with X, B, or A registers so that all read flags must be tested. Table 5-4 lists units and read flags tested for all clear conditions.

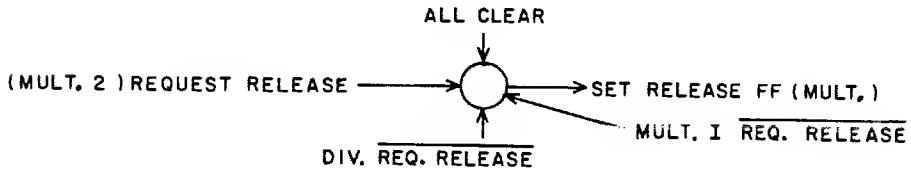
FUNCTION UNIT	READ FLAGS		
	X	B	A
ADD	X		
LONG ADD	X		
MULTIPLY 1	X		
MULTIPLY 2	X		
DIVIDE	X		
BOOLEAN	X		
SHIFT	X	X	
INCREMENT 1	X	X	X
INCREMENT 2	X	X	X

Table 5-4

DATA TRUNK PRIORITY

a) Go Store Priority

After the Request Release FF of a given unit is set, an AND gate with All Clear, Trunk Priority and Request Release input will allow the Unit Release FF to be set, sending out a Go Store signal e.g., the AND gate of the Multiply 2 Request Release looks like this:



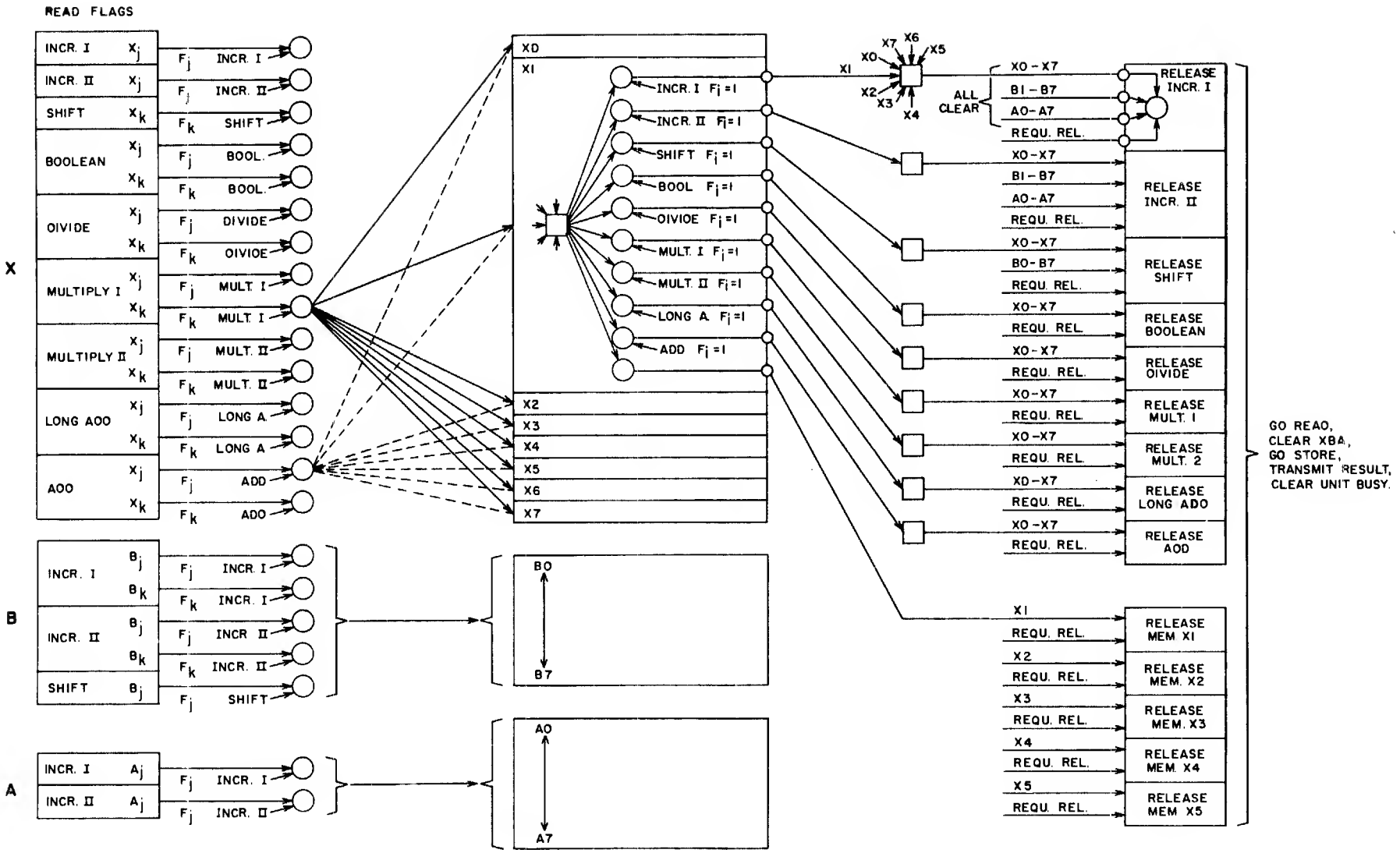
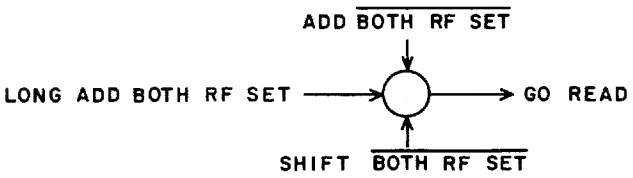


Figure 5-8. All Clear

This means that the Multiply 2 unit (Priority 3) can only be released if Divide unit and Multiply I unit are not requesting release. An exception is the Increment I and II units which have the priority gate at the output of the Release FF (I32); (see Set Read Flags block diagram, Figure 5-6)

b) Go Read Priority

The outputs of the Read Flags of a given unit are ANDed together with the output of the Read Flags of another unit e.g.



DATA TRUNK PRIORITIES

GO read	GO store	
1. ADD	1. SHIFT	TRUNK 1
2. SHIFT	2. ADD	
3. LONG ADD	3. LONG ADD	
1. DIVIDE	1. BOOLEAN	TRUNK 2
2. MULTIPLY 1	2. DIVIDE	
3. MULTIPLY 2	3. MULTIPLY 1	
4. BOOLEAN	4. MULTIPLY 2	
1. INCREMENT 1	1. INCREMENT 1	TRUNK 3
2. INCREMENT 2	2. INCREMENT 2	

Table 5-5

GO STORE/GO READ

The following discussion refers to the block diagram in Figure 5-10. Assume that Functional Unit X is sending a Request Release setting its Request Release FF. If the All clear and Go Store Priority is made, the Unit Release FF will be set.

The outputs of the Unit Release FF will do the following:

1. Clear Request Release FF
2. Clear Unit Busy FF
3. Enable the F_i (Result) designator to clear its respective XBA designator.
4. A "go store" signal enables the contents of the F_i register into the Entry Control specifying the register into which the result of the releasing unit must be stored. Another "Go store" signal is sent directly to the Entry control to allow it to recognize the F_1 input.
5. A "transmit" signal is sent to the Functional Unit telling it to release its result.

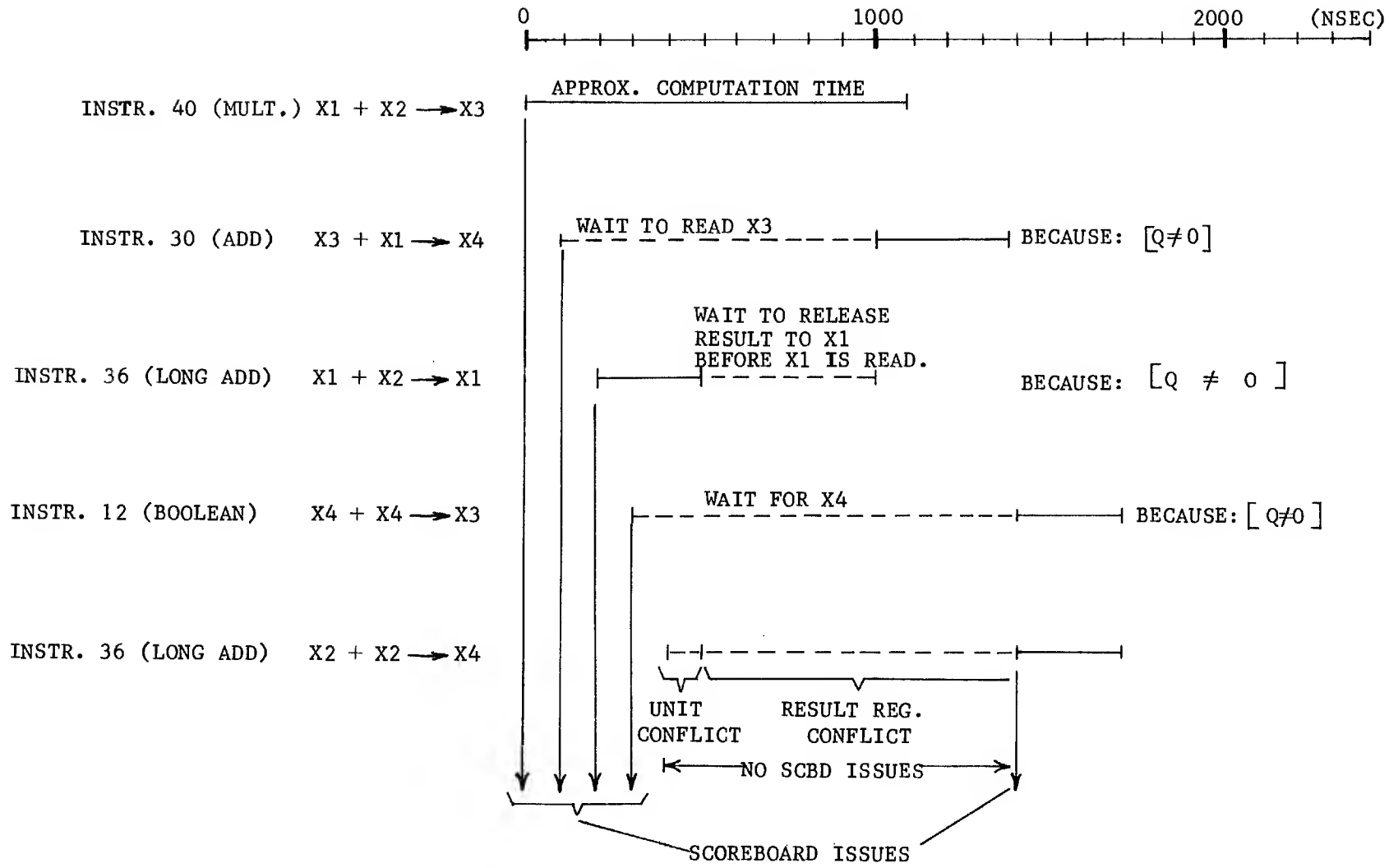
Once the releasing unit has stored its result it is free to start a new computation. If there are units waiting to read this result register after the result is stored, their Q designator(s) will contain the unit code of the releasing unit.

These Q designators would then set their respective Read Flags. If both Read Flags are now set for the waiting unit, its go read priority is checked. If there is more than one waiting unit with all its read flags set, the priority gate will allow the unit with highest priority to go and read its operands as follows:

a) Clear Read Flags
b) Set Go Read FF to inhibit entrance to the read flags during the read sequence.
c) Send an enable to the outputs of the F_j and Fk designators to transfer the contents to the Exit Control of the Operating Regs.
d) Send a "Go Read" to the Exit Control to allow it to recognize the register numbers being sent by the F designators.
e) Send a "Go F.U." to allow the unit to accept the incoming operands.

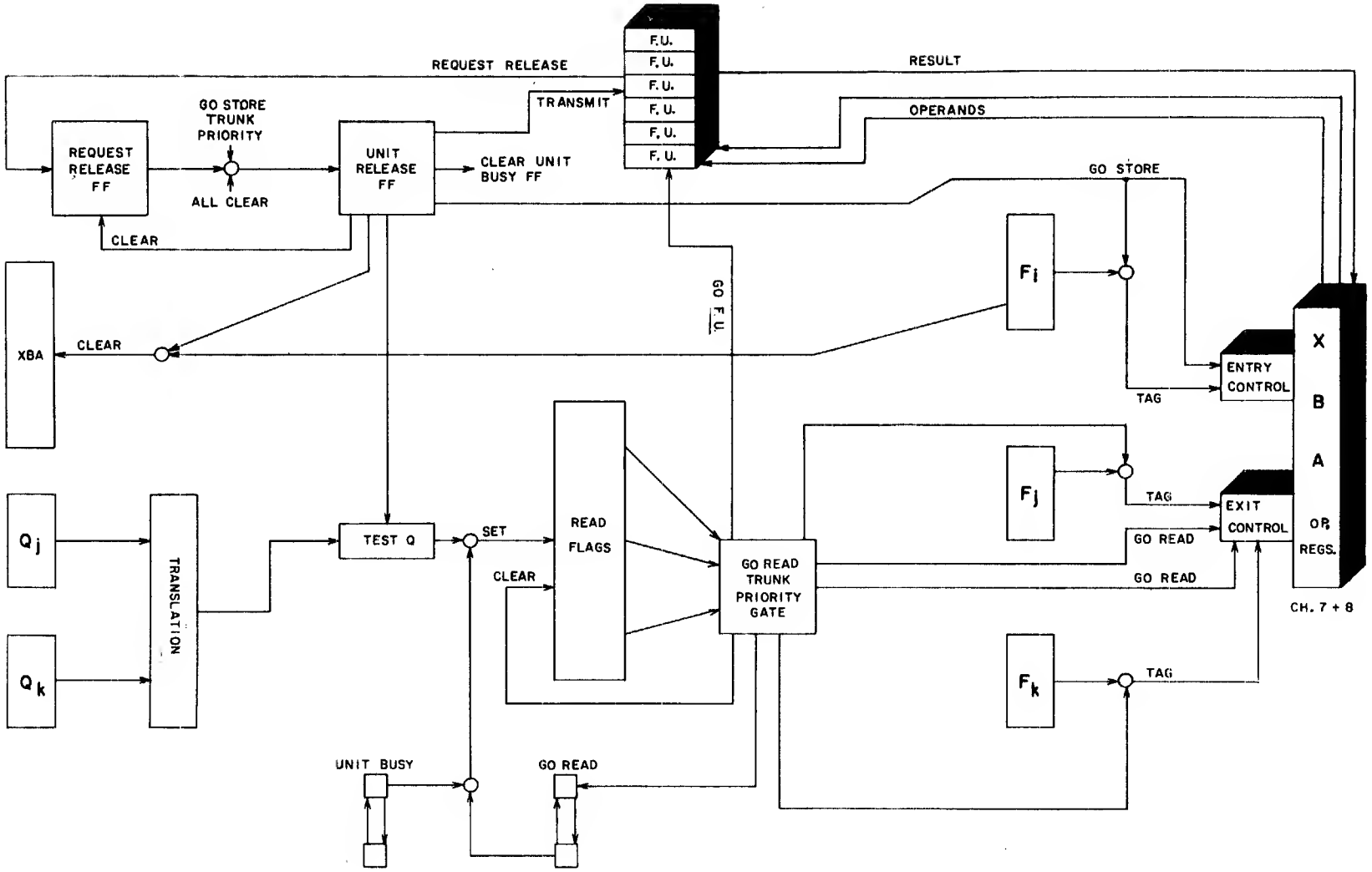
Waiting units can therefore be identified by the set state of their Busy FF and a cleared Go Read FF.

Note that the time elapse from a unit's request release to placing the result on the trunk may be within a minor cycle when the all clear is present at request release time and trunk priority favors the requesting unit.



APPROX. INSTRUCTION EXECUTION TIMING

Figure 5-9



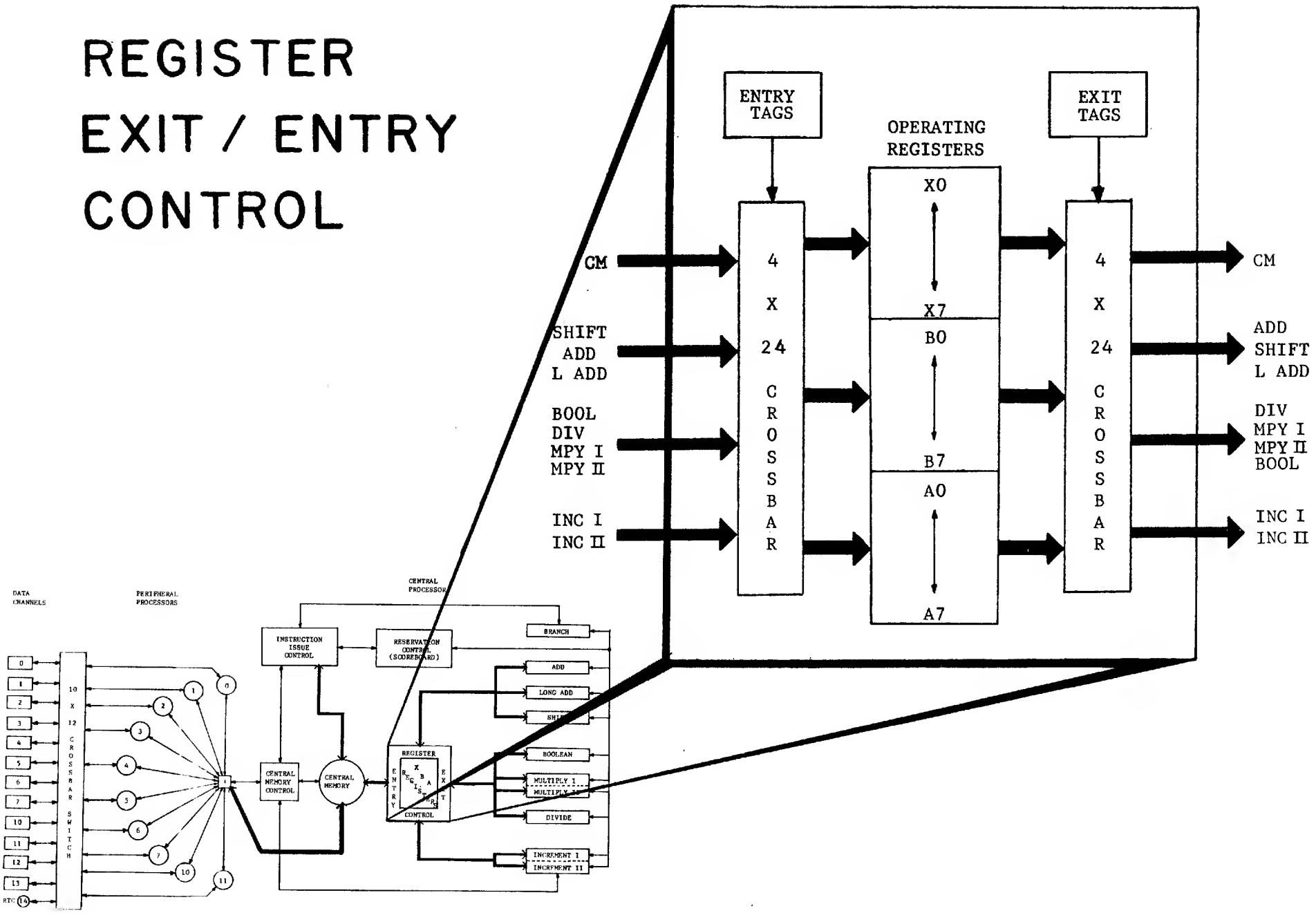
REQUEST RELEASE BLOCK DIAGRAM

Figure 5-10

CHAPTER VI

EXIT/ENTRY CONTROL AND DATA TRUNKS

REGISTER EXIT / ENTRY CONTROL



CHAPTER VI

ENTRY/EXIT CONTROL AND DATA TRUNKS

ENTRY CONTROL

FOR X REGISTERS

The data from the different F.U.s and Central Memory are received by the Input Registers on Chassis 7 and 8 (e.g. Ch. 7: A37, B37, C37, H37, etc. Ch. 8: A01-08, B01-05, D01, etc.)

The "Go Store" signal will enable the contents of the F designator into the Translator of the Entry Control to decide into which register the data is to be stored (Refer to Figure 6-1). For the Increment Unit the Input Reg only receives 18 bits, therefore the 2^{17} bit (Sign Bit) is used to fill up the bit locations 2^{18}-2^{59} in the X register (Sign Extension). For those instructions that need a memory reference, the address tags are taken from the Hopper and translated to gate the contents of the Input Reg into one of five Buffer Register D1-D5 correcponding to X1-X5. Another translation of these tags, together with an Accept will allow the setting of one of the Request Release FF for Memory Ch. 1-5. Once the All Clear gate is made, the Release FF for the selected Memory Channel will be set, allowing the setting of the corresponding D-X FF. The data is now gated from the Buffer Registers into the X operating registers. Bits 2^{0}-2^{35} of the X registers are on Ch. 7, the 2^{36}-2^{59} on Ch. 8.

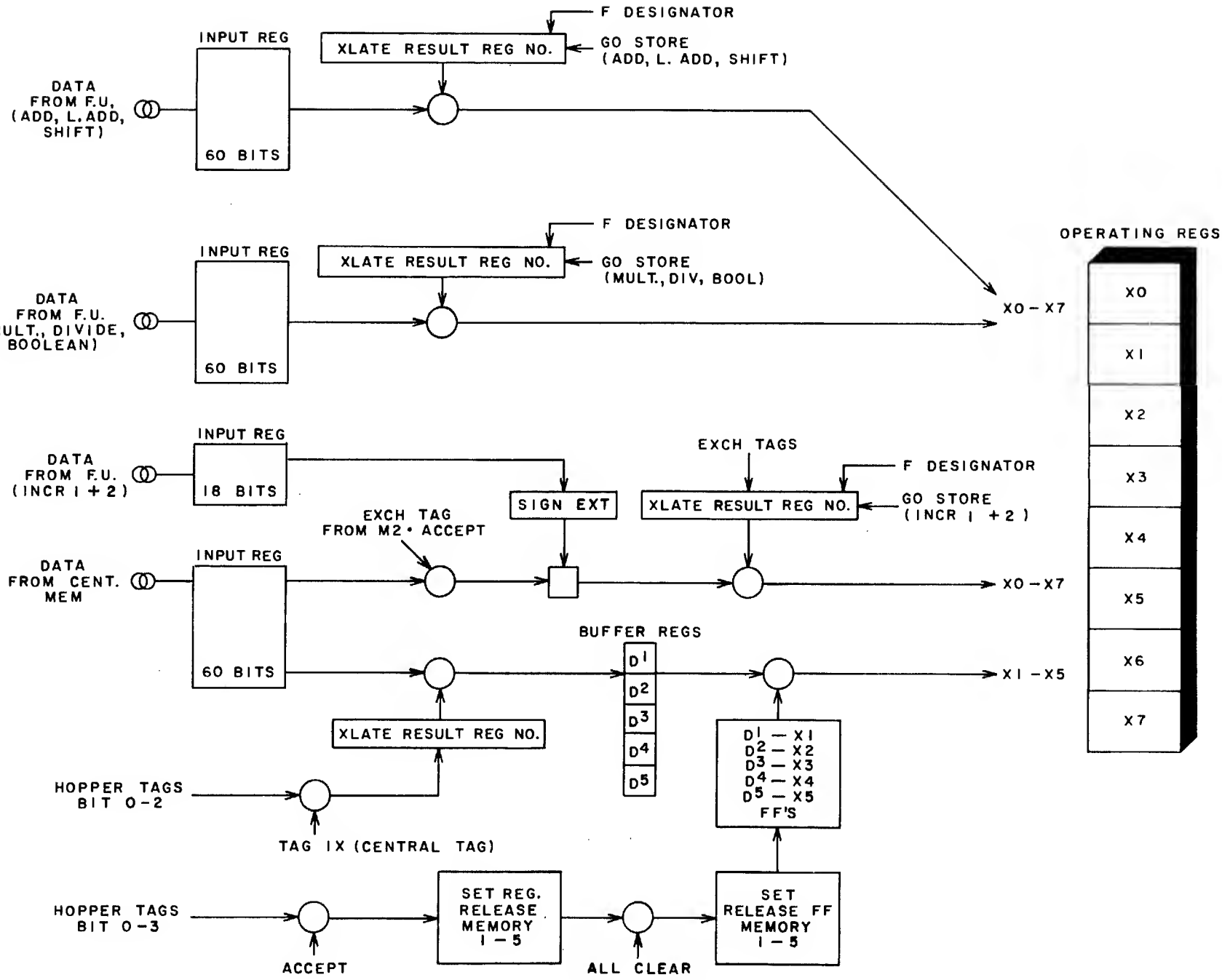
For an Exchange Jump, the tags from the Exchange Tag Control and the Exchange Tag (1 bit) from the Hopper (M2) together with an "accept" will gate the data from memory into its respective X register.

FOR A, B REGISTERS

The same applies here that has been said about the X registers (see Figure 6-2) except that A and B registers contain only 18 bits.

EXIT CONTROL

The "Go Read" signal from the Test Trunk Priority network will enable the contents of the F_j- and F_k- designators into the Translators of the Exit Control. This specifies the transfer from a particular X, B or A operating register to a particular functional unit. The $F_{j,k}$-designators are combined into 9 different groups. These can be seen in the block diagram of the Exit Control (Figure 6-3)



Entry Control - X Regs. Figure 6-1

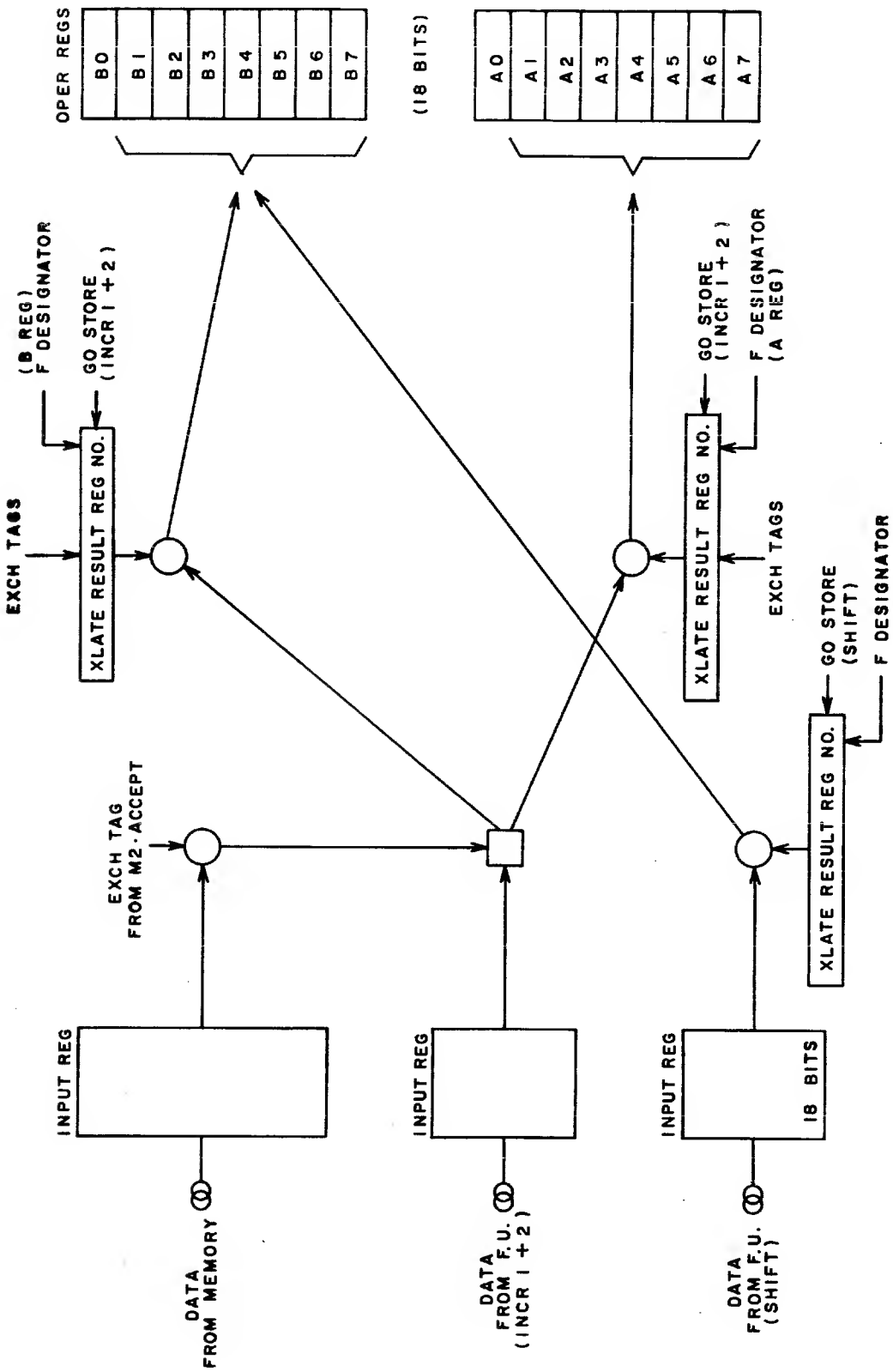


Figure 6-2. Entry Control - B, A Regs.

The $F_{j,k}$-designators from the same group will use the same Network to gate the contents of the XBA Operating Register into the F.U. However, in every group there is also a "go" bit from the Test Trunk Priority network. These bits will help gate the data to the proper part of the Output Network.

The Output Network and the data trunks to the functional units are grouped to correspond generally with the $F_{j,k}$ designator groupings (e.g., Operands 1 of Divide, Multiply 1, Multiply 2 and Boolean use the same data trunk).

During an Exchange Jump, the contents of the XBA Register are transmitted to Central Memory. The tag that controls this transfer comes from the Hopper M4 of the Stunt Box. Only the lower 4 bits of the Hopper tag are sent to the Exit Control network.

The 3 lower bits are translated to determine which of the 7 X, B or A Registers will be gated to Central Memory. When the 2^3 bit is zero (6X) the contents of the B-Operating Registers are gated into the lower 18 bits of the Output network going to the Store Distributer. Simultaneously the contents of the A-Operating Registers are gated into bit positions 2^{18}-2^{35} of the Output Network. When the 2^3bit is one (7X) the contents of the X-Registers will be gated to Central Memory via the Output network.

For a Central Processor Write (X6 $\longrightarrow$ CM or X7 $\longrightarrow$ CM) a situation exits which is similar to that described previously for Exchange Jump. The Hopper tag (16 or 17) will gate the contents of Operating Register X6 or X7 to Central Memory via the Output network.

Note: The B and A registers are located on chassis 7 along with bits 0-35 of the X registers. Bits 36-59 of the X registers are located on chassis 8. Therefore, all tags and designators must be sent to both chassis 7 and 8.

DATA TRUNKS

Four Data Trunks (shown in Figure 6-5) transmit the 2 Operands from the output network of the XBA-Operating Register to the proper functional unit. Data is also transferred on trunks to Central Memory. Four other Data Trunks transmit the result from the functionl trunk and data from Central Memory to the Input Register of XBA-Operating register.

Data Trunk 1 connects the output network of the XBA-Register to the Long Add, Add and Shift functional units. There are 60 bits of coaxial cable for transmitting Operand 1 and 60 bits of cable, for Operand 2.

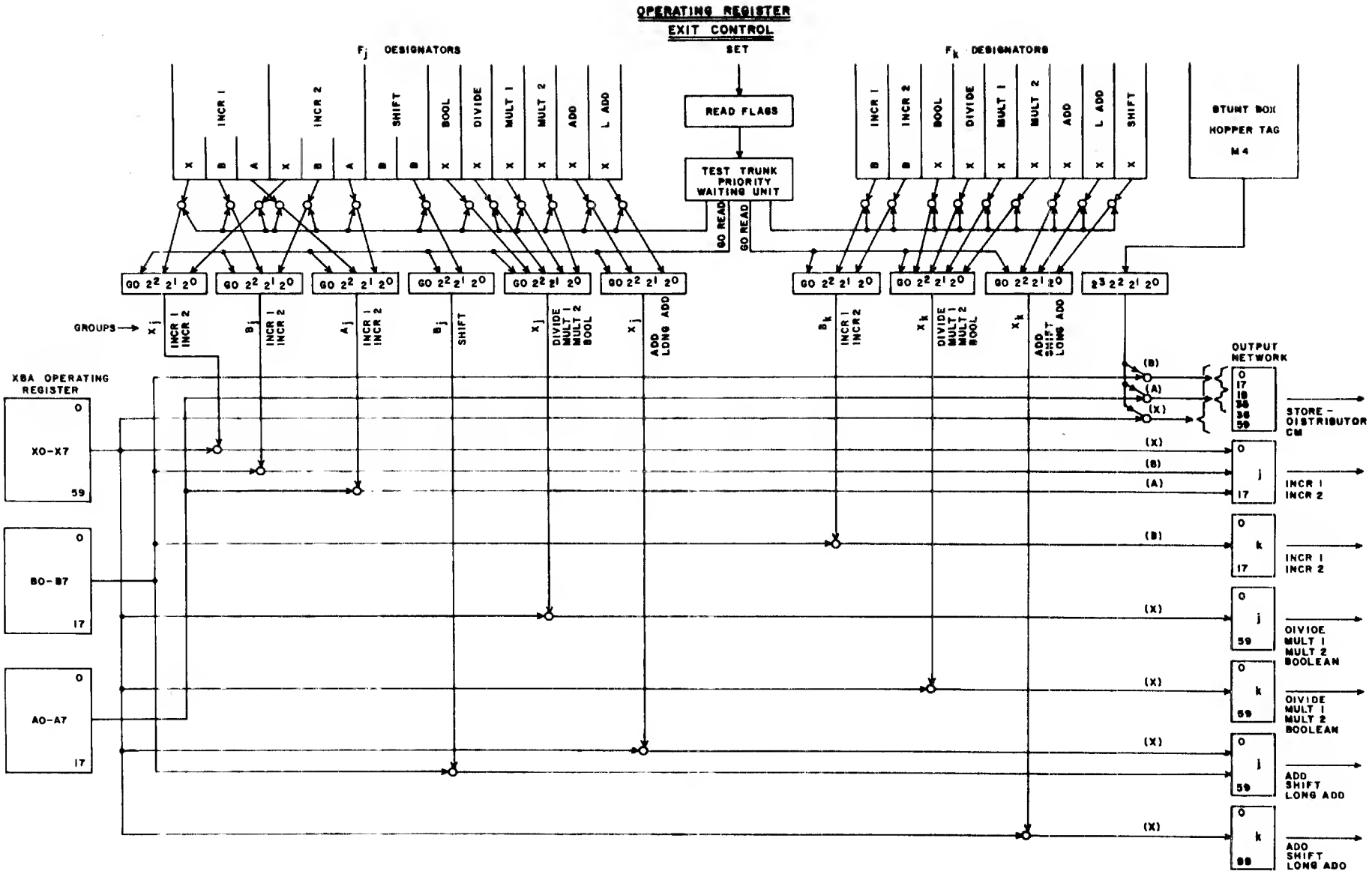

OPERATING REGISTER
EXIT CONTROL
SET
F_j DESIGNATORS
F_k DESIGNATORS
INCR 1
INCR 2
SHIFT
BOOL
DIVIDE
MULT 1
MULT 2
ADD
L ADD
READ FLAGS
TEST TRUNK PRIORITY WAITING UNIT
GO READ
STUNT BOX
HOPPER TAG
M4
GROUPS
XBA OPERATING REGISTER
X0-X7
B0-B7
A0-A7
OUTPUT NETWORK
STORE - DISTRIBUTOR CM
INCR 1 INCR 2
DIVIDE MULT 1 MULT 2 BOOLEAN
ADD SHIFT LONG ADD

Figure 6-3

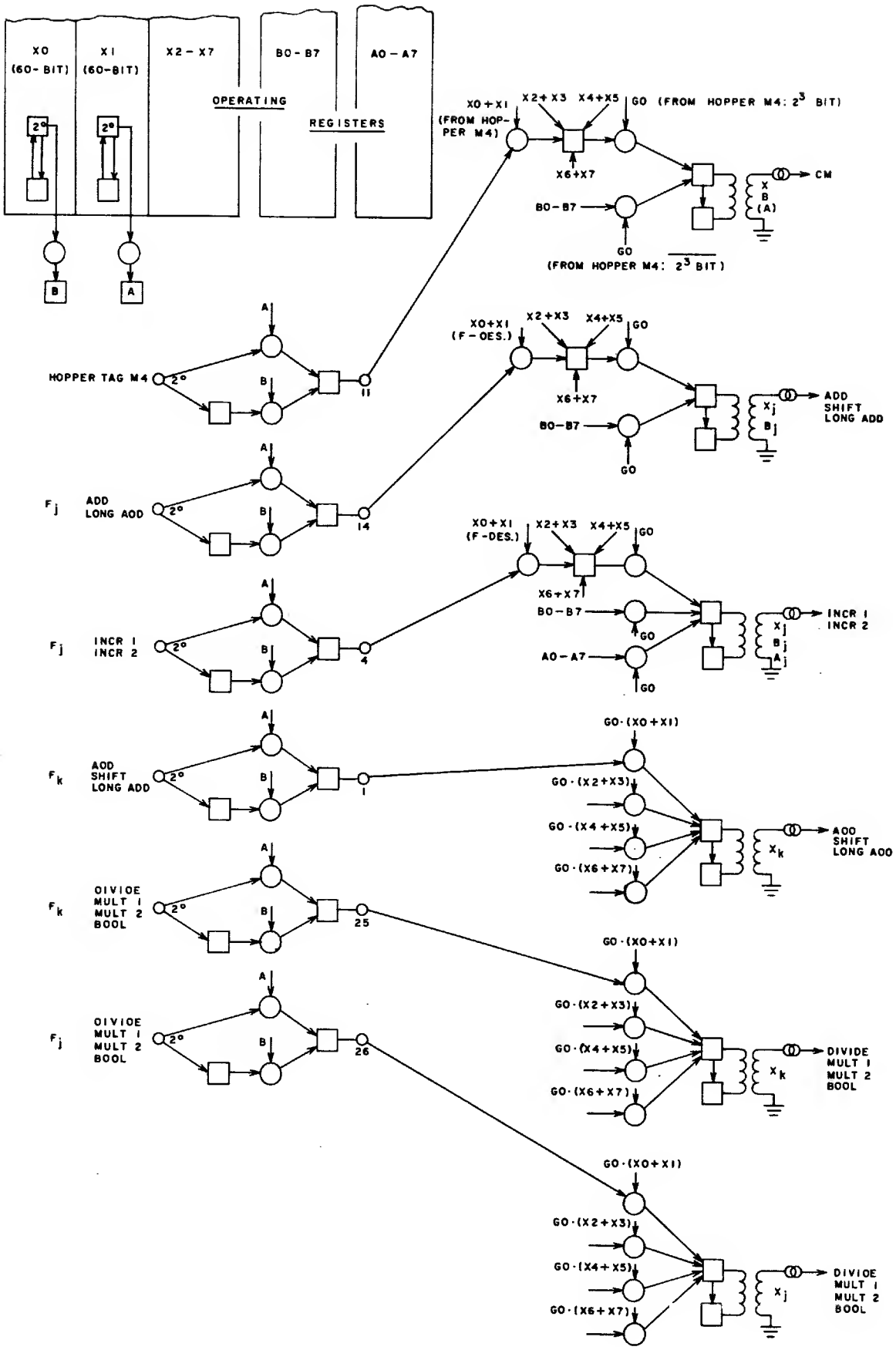


Figure 6-4. Operating Reg. Exit Control

Data Trunk 2 connects the output network of the XBA-Registers to the Divide, Boolean, Multiply 1 and Multiply 2 functional units. The lower 48-bits of the operands go via chassis 6 (Multiply 1 and 2) to chassis 2 (Divide and Boolean). The upper 12-bits go directly to chassis 2 from EXIT Control and are not sent to chassis 6 (except bit 2^{59}) which contains only the Multiply coefficient logic.

The results coming from the functional units and the data coming from Central Memory will be transmitted by Data Trunks 1', 2' 3', and 4' to the Input Register of the Entry Control network.

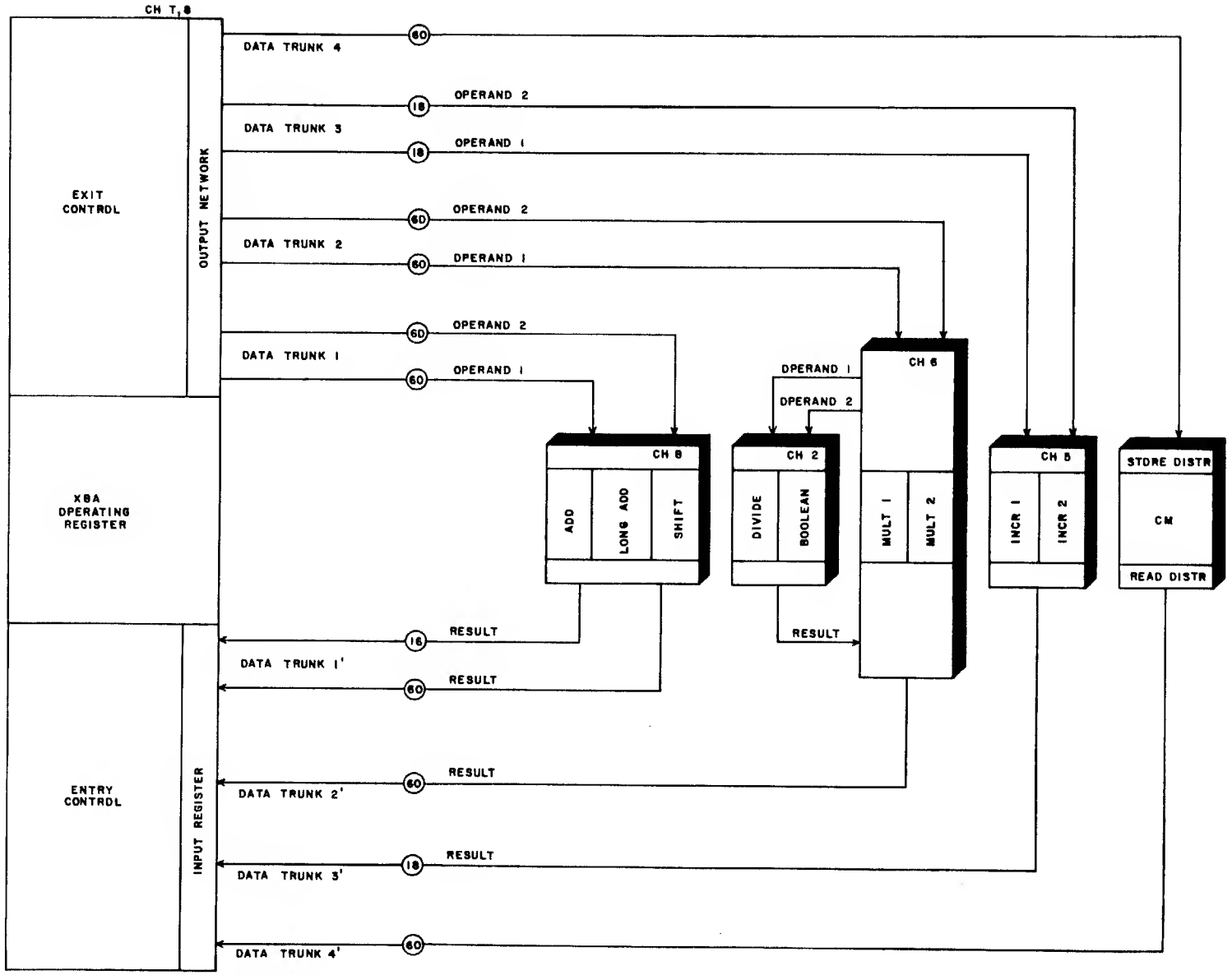


Figure 6-5. Data Trunks

APPENDIX A

CENTRAL PROCESSOR TIMING NOTES

6600 CPU TIMING NOTES

1. The times given in the reference manual are computational times - the time needed after the execution start until the result is computed and ready to be stored into the result register. Times are given in minor cycles (1 minor cycle = 100 nanoseconds).

2. A functional unit can not be reissued until one minor cycle after it has been released. (Result is stored by Entry Control during the minor cycle after release)

3. A result register value may be used as an operand to another instruction as soon as the result has been stored into the register (same minor cycle). This result register will not be freed for use as a result register of another instruction until one cycle after the result has been stored into that register. (No trunk priority is considered.)

4. Instructions are issued to the functional units if:

   a. The word containing the instruction is in the stack.
   b. The functional unit(s) needed are free, and
   c. The result register(s) needed are free.

   If these conditions are not met, all further instruction issues are held until they are satisfied. Each issued 15-bit instruction requires one minor cycle before the next instruction is available for issue. Each issued 30-bit instruction requires two minor cycles before the next instruction is available for issue.

5. Execution within a functional unit does not start until the operand(s) are available. The two operands required are fetched from the registers at the same time (one operand is not loaded while the unit waits for the second operand).

6. In instructions 02-07, where more than one functional unit is used, the instruction is not issued until both functional units involved are free.

7. Times given for instructions 01-07 and 50-57 do not consider any memory conflict conditions.

8. In instructions 50-57, if i=1, 2 ... 5 (load from memory instructions), the Xi register value is not available until 8 minor cycles after the start of the instruction execution (assuming no memory conflicts). When two load instructions begin execution one minor cycle apart at least one extra

minor cycle is required for execution of the later instruction. Therefore the second executed instruction would require 9 cycles for the load, 4 cycles for the increment unit and 4 cycles for the A register.

9. In instructions 50-57, if i = 6 or 7 (store to memory instructions), the Xi register is not available for a result register until 8 minor cycles after the instruction begins execution (assuming no memory conflicts). When two store instructions begin execution one minor cycle apart, one extra cycle is required for execution of the later instruction. Therefore, the second executed instruction would require 9 cycles for the store, 4 cycles for the increment unit and 4 cycles for the A register. A store instruction checks the X register before being issued. The X register is available as an entry operand register while the store is taking place.

10. When executing sequential instructions that are not in the stack, the minimum time is one word of instructions every 8 cycles. The time of issue of the last parcel of an instruction word to the time of issue of the first parcel of the next instruction word (while executing sequential instructions that are not in the stack) requires a minimum of 4 cycles. If the last instruction in a word is a 30-bit instruction, a minimum of 5 cycles are required from the time of issue of this instruction to the time of issue of the first instruction of the next word.

11. All 03 branches made within the stack require 9 minor cycles. An 03 branch to the next sequential word is recognized as a branch within the stack and requires 9 minor cycles.

12. When a branch out of the stack is taken, 15 minor cycles are normally required for an 03ijK instruction and 14 minor cycles for other branch instructions (considering no memory conflicts), timed from the start of the branch instruction execution to the availability of the branched-to word instruction to a functional unit (instruction ready for issue).

13. Eleven cycles are required for the 03ijK instructions when the branch is not taken (time from branch execution to issue of the next instruction) if in the stack or if falling through to an instruction within the same word. Out of stack fall-through to the next word takes 14 cycles.

14. Ten cycles are required for 04ijK - 07ijK instructions when the branch is not taken (time from branch execution to issue of the next instruction) if in the stack or falling through to an instruction within the same word. Out of stack fall-through to the next word takes 13 cycles.

15. The B0 register is handled like any other Bi register for timing purposes (i.e., B0 will hold up execution of an instruction if it is a result register of a previous, non-completed instruction, etc.).

16. Neither increment unit may be involved in a load operation if a store operation is to be issued, and neither increment unit may be involved in a store operation if a load operation is to be issued. The sequential loading of instruction words does not affect the load/store conditions of the increment units.

17. The operand registers are available to more than one functional unit in the same minor cycles if the units are in different groups.

GROUP 1	GROUP 2	GROUP 3
Divide	Add	Increment 1
Multiply 1	Shift	Increment 2
Multiply 2	Long Add	
Boolean		

18. The time needed for a functional unit to operate on indefinite, out-of-range or zero values is the same as for normal, in-range values (i.e., no gain or loss in execution time due to a unit recognizing an indefinite operand and setting an indefinite result).

19. An index jump instruction (02) will always destroy the stack. If an unconditional jump backward in the stack is desired, an 0400K instruction should be used (to save memory access time for instructions).

20. A return jump instruction (01) will always destroy the stack.

21. After a result has been computed by a functional unit, the result register is checked to see if it has been reserved as an operand register (for a previously issued instruction). This is done so that the result will not overlay an operand to a previously issued instruction. If a unit (#1) is waiting for an operand to be fetched by another unit (#2) before storing its result, for timing considerations:

    a. The result register is available to a third unit (#3) as an operand, the cycle following the fetch, and

    b. The register may be available as a result register two cycles following the fetch, and

    c. The unit is freed two cycles following the fetch.

APPENDIX B

NON-STANDARD OPERAND FORMS

NON-STANDARD FLOATING POINT ARITHMETIC

The following is a tabulation of operations (Add, Subtract, Multiply, Divide) using various combinations of operands to supplement Table 3-3 (page 3-13). The key to operands and results used in the table is as follows:

KEY:

OPERANDS			RESULTS		
+0	=	0000 X...X	0	=	0000 0...0
-0	=	7777 X...X	IND	=	1777 0...0
+ ∞	=	3777 X...X	+ ∞	=	3777 0...0
- ∞	=	4000 X...X	- ∞	=	4000 0...0
+IND	=	1777 X...X			
-IND	=	6000 X...X			
W	=	Any word except ± ∞ , ±IND			
N	=	Any word except ± ∞ , ±IND, or ±0			

ADD

Xi=Xj+Xk

(Instructions 30, 32, 34)

		Xk			
		W	+ ∞	- ∞	±IND
Xj	W	-	+ ∞	- ∞	IND
	+ ∞		+ ∞	IND	IND
	- ∞		∞	- ∞	IND
	±IND				IND

SUBTRACT

Xi=Xj-Xk

(Instructions 31, 33, 35)

		Xk			
		W	+ ∞	- ∞	±IND
Xj	W	-	- ∞	+ ∞	IND
	+ ∞	+ ∞	IND	+ ∞	IND
	- ∞	- ∞	- ∞	IND	IND
	±IND	IND	IND	IND	IND

MULTIPLY

Xi=Xj*Xk

(Instructions 40, 41, 42)

		Xk						
		+N	-N	+0	-0	+∞	-∞	±IND
Xj	+N	-	-	0	0	+∞	-∞	IND
	-N		-	0	0	-∞	+∞	IND
	+0			0	0	IND	IND	IND
	-0				0	IND	IND	IND
	+∞					+∞	-∞	IND
	-∞						+∞	IND
	±IND							IND

DIVIDE

Xi=Xj/Xk

(Instructions 44, 45)

		Xk						
		+N	-N	+0	-0	+∞	-∞	±IND
Xj	+N	-	-	+∞	-∞	0	0	IND
	-N	-	-	-∞	+∞	0	0	IND
	+0	0	0	IND	IND	0	0	IND
	-0	0	0	IND	IND	0	0	IND
	+∞	+∞	-∞	+∞	-∞	IND	IND	IND
	-∞	-∞	+∞	-∞	+∞	IND	IND	IND
	±IND	IND	IND	IND	IND	IND	IND	IND

COMMENT SHEET

6600 CENTRAL PROCESSOR, Volume I

Publication No. 020167

FROM: Name: _____

Address: _____

COMMENTS: (Describe errors, suggested additoins or deletions, and include page numbers, etc.)

CONTROL DATA INSTITUTE
3255 Hennepin Avenue So.
Minneapolis, Minnesota 55408
612-827-4715

CONTROL DATA INSTITUTE
5630 Arbor Vitae Street
Los Angeles, California 90045
213-670-3640

CONTROL DATA INSTITUTE
3717 Columbia Pike
Arlington, Virginia 22204
703-521-3700